Contents

Preface ............................................................................................................................................................ xiii
Audience ........................................................................................................................................................ xiii
Documentation Accessibility ....................................................................................................................... xiii
Related Documents ................................................................................................................................ xiv
Conventions ............................................................................................................................................... xiv

What’s New in Oracle BPEL Process Manager? ...................................................................................... xxvii

Part I Introduction and Concepts

1 Introduction to Oracle BPEL Process Manager

What Is BPEL? ................................................................. 1-1
What Is Oracle JDeveloper? .................................................. 1-3
How to Use This Guide .................................................... 1-4
Getting Started with Demonstrations, Activity and Conceptual References, and Tutorials........... 1-6
  What Demonstrations Are Available? ........................................ 1-7
  What Activity and Conceptual References Are Available? ....... 1-9
  What Tutorials Are Available? .............................................. 1-10
Summary ................................................................................ 1-15

2 Getting Started with Oracle BPEL Process Manager

Overview of Oracle BPEL Process Manager Components ............................................................... 2-1
Starting Oracle BPEL Process Manager Components ...................................................................... 2-2
Overview of the BPEL Designer Environment .................................................................................... 2-3
  Overview of BPEL Project Creation and Oracle JDeveloper .......................................................... 2-3
  Application Navigator ................................................................. 2-6
  Diagram Window .................................................................. 2-7
  Source Window ...................................................................... 2-8
  History Window .............................................................. 2-10
  Component Palette .............................................................. 2-10
  Property Inspector ................................................................. 2-12
  Structure Window ................................................................. 2-12
  Log Window ........................................................................... 2-13
  Editing Project Files in Oracle JDeveloper .......................... 2-14
Part II  Reviewing Key BPEL Development Concepts and Code Samples

3  Manipulating XML Data in BPEL

Use Cases for Manipulating XML Data in BPEL ................................................................. 3-1
Overview of Manipulating XML Data in BPEL Concepts .................................................. 3-2
How XML Data Works in BPEL ...................................................................................... 3-2
About Data Manipulation and XPath Standards ............................................................... 3-2
Initializing a Variable with Expression Constants or Literal XML ................................. 3-4
Copying Between Variables ............................................................................................. 3-5
Accessing Fields Within Element-Based and Message Type-Based Variables ............... 3-6
Assigning Numeric Values ............................................................................................... 3-6
Mathematical Calculations with XPath Standards ......................................................... 3-7
Assigning String Literals ................................................................................................. 3-7
Concatenating Strings ..................................................................................................... 3-7
Assigning Boolean Values .............................................................................................. 3-8
Assigning Date or Time .................................................................................................... 3-8
Manipulating Attributes ................................................................................................. 3-9
Manipulating XML Data with bpelx Extensions ............................................................... 3-10
  bpelx:append ................................................................................................................ 3-10
  bpelx:insertBefore ...................................................................................................... 3-11
  bpelx:insertAfter ....................................................................................................... 3-12
  bpelx:remove ............................................................................................................. 3-12
  bpelx:rename and XSD Type Casting ......................................................................... 3-13
  bpelx:copyList ........................................................................................................... 3-13
Validating XML Data with bpelx:validate ..................................................................... 3-16
Manipulating XML Data Sequences That Use Arrays ...................................................... 3-16
  Statically Indexing into an XML Data Sequence That Uses Arrays ............................. 3-16
  Determining Sequence Size ....................................................................................... 3-17
  Dynamically Indexing by Applying a Trailing XPath to an Expression ...................... 3-17
    Dynamic Indexing Example .................................................................................... 3-18
  Using the bpelx:append Extension to Append New Items to a Sequence ................... 3-18
  Merging Data Sequences ............................................................................................ 3-19
  Dynamically Indexing with the BPEL getElement Function ....................................... 3-19
  Generating Functionality Equivalent to an Array of an Empty Element ...................... 3-20
  SOAP-Encoded Arrays Not Supported ..................................................................... 3-21
Converting from a String to an XML Element ................................................................. 3-21
Differences Between Document-Style and RPC-Style WSDL Files ................................. 3-22
Adding a Custom WSIF Provider ..................................................................................... 3-23
Task 1: Register the WSDL Extension to the WSDL Reader .......................................... 3-23
Step 1: Creating a Project ................................................................. 5-15
Step 2: Configuring Partner Links and File Adapter Services 5-15
   Creating an Initial Partner Link and File Adapter Service 5-15
   Creating a Second Partner Link and File Adapter Service 5-17
   Creating a Third Partner Link and File Adapter Service 5-18
Step 3: Creating Three Receive Activities ................................. 5-19
   Creating an Initial Receive Activity .................................................. 5-19
   Creating a Second Receive Activity ................................................ 5-20
   Creating a Third Receive Activity ................................................. 5-20
Step 4: Creating Correlation Sets .............................................. 5-21
   Creating an Initial Correlation Set .................................................. 5-21
   Creating a Second Correlation Set ................................................ 5-21
Step 5: Associating Correlation Sets with Receive Activities .... 5-22
   Associating the First Correlation Set with a Receive Activity 5-22
   Associating the Second Correlation Set with a Receive Activity 5-22
   Associating the Third Correlation Set with a Receive Activity 5-23
Step 6: Creating Property Aliases .............................................. 5-24
   Creating Property Aliases for NameCorr ........................................ 5-24
   Creating Property Aliases for IDCorr ........................................... 5-24
Step 7: Reviewing WSDL File Content .................................... 5-25
Summary .................................................................................... 5-26

6 Parallel Flow
Use Case for Parallel Flows ......................................................... 6-1
Overview of Parallel Flow Concepts ......................................... 6-1
Defining a Parallel Flow ............................................................ 6-2
Customizing the Number of Flow Activities by Using the flowN Activity 6-4
   BPEL Code Example of the flowN Activity .................................... 6-6
Summary .................................................................................... 6-7

7 Conditional Branching
Use Case for Conditional Branching ........................................ 7-1
Overview of Conditional Branching Concepts .......................... 7-1
Using a While Activity to Define Conditional Branching ............ 7-2
Summary .................................................................................... 7-5

8 Fault Handling
Use Case for Fault Handling ....................................................... 8-1
Defining a Fault Handler ......................................................... 8-1
BPEL Standard Faults ............................................................... 8-3
   Categories of BPEL Faults ......................................................... 8-3
   Business Faults ................................................................. 8-3
   Run-time Faults ............................................................... 8-3
      bindingFault .......................................................... 8-4
      remoteFault ......................................................... 8-5
12 Interaction Patterns

One-Way Message ................................................................. 12-1
Synchronous Interaction ....................................................... 12-2
Asynchronous Interaction ...................................................... 12-3
Asynchronous Interaction with Timeout ................................. 12-4
Asynchronous Interaction with a Notification Timer ................ 12-5
One Request, Multiple Responses ........................................... 12-6
One Request, One of Two Possible Responses ......................... 12-7
One Request, a Mandatory Response, and an Optional Response 12-8
Partial Processing ................................................................. 12-9
Multiple Application Interactions ............................................ 12-10
Summary ................................................................................ 12-11

13 XSLT Mapper and Transformations

Use Case for Transformation .................................................. 13-1
Creating an XSL Map File ...................................................... 13-1
Creating a New XSL Map File ................................................... 13-2
Creating an XSL Map File from Imported Source and Target Schema Files 13-3
Overview of the XSLT Mapper .................................................. 13-5
Notes on the Mapper ............................................................. 13-6
Using the XSLT Mapper .......................................................... 13-7
Simple Copy by Linking Nodes .................................................. 13-7
Setting Constant Values ......................................................... 13-7
Adding Functions .................................................................... 13-8
Editing Function Parameters .................................................... 13-9
Chaining Functions .................................................................. 13-10
Named Templates ..................................................................... 13-10
Importing User-Defined Functions .......................................... 13-10
Editing XPath Expressions ...................................................... 13-11
Adding XSLT Constructs .......................................................... 13-12
Conditional Processing with xsl:if ........................................... 13-12
Conditional Processing with xsl:choose ................................... 13-13
Handling Repetition or Arrays .................................................. 13-14
Automatically Mapping Nodes ............................................... 13-15
Auto Map with Confirmation .................................................. 13-17
Viewing Unmapped Target Nodes .......................................... 13-18
Generating Dictionaries .......................................................... 13-19
Create a Human Task Definition with the Human Task Editor ........................................ 15-12
Associate the Human Task Definition with a BPEL Process ........................................ 15-12
Generate the Task Display Form .................................................................................. 15-13

Task 1: Creating the Human Task Definition with the Human Task Editor ........................................ 15-13
Accessing the Human Task Editor .................................................................................. 15-13
  From the Application Navigator .................................................................................. 15-13
  From the Component Palette ...................................................................................... 15-14
Reviewing the Sections of the Human Task Editor ...................................................... 15-14
Specifying the Task Title, Priority, Outcome, and Owner .............................................. 15-15
  Specifying a Task Title and Priority ........................................................................... 15-16
  Specifying a Task Outcome ....................................................................................... 15-16
  Specifying a Task Owner ............................................................................................ 15-18
Specifying the Task Payload Data Structure .................................................................. 15-21
Assigning Task Participants ............................................................................................ 15-22
  Specifying Task Approvers ...................................................................................... 15-23
  Configuring the Single Approver Participant Type ...................................................... 15-24
  Configuring the Group Vote Participant Type .......................................................... 15-27
  Configuring the Management Chain Participant Type ................................................. 15-30
  Configuring the Sequential List of Approvers Participant Type ................................... 15-32
  Configuring the FYI Assignee Participant Type .......................................................... 15-35
  Configuring the External Routing Service Participant Type ....................................... 15-36
  Allowing All Participants to Invite Other Participants ................................................. 15-37
Abruptly Completing a Condition .................................................................................. 15-38
Escalating, Renewing, or Ending the Task ...................................................................... 15-39
  Overview or Escalation and Expiration Policy ........................................................... 15-39
  Never Expire Policy .................................................................................................. 15-41
  Expire After Policy ................................................................................................... 15-41
  Renew After Policy ................................................................................................... 15-42
  Escalate After Policy .................................................................................................. 15-42
Specifying Participant Notification Preferences ............................................................. 15-43
  Notifying Recipients of Changes to Task Status ......................................................... 15-44
  Editing the Notification Message .............................................................................. 15-45
  Setting Up Reminders ............................................................................................... 15-46
Specifying Advanced Settings ........................................................................................ 15-47
  Specifying Escalation Rules ....................................................................................... 15-47
  Specifying WordML Style Sheets for Attachments .................................................... 15-48
  Specifying Style Sheets for Attachments ................................................................... 15-48
  Specifying Multilingual Settings ............................................................................... 15-48
Overriding Default System Actions .............................................................................. 15-49
Overriding Default Exception Management ................................................................... 15-50
Specifying Callback Classes on Task Status ................................................................ 15-51
Allowing Task and Routing Customization in BPEL Callbacks ..................................... 15-52
Exiting the Human Task Editor and Saving Your Changes ............................................ 15-53

Task 2: Associating the Human Task with a BPEL Process ............................................. 15-53
  Associating a Human Worklist Task with a BPEL Process ......................................... 15-54
  Opening a Human Task Activity Already Associated with a BPEL Process ............... 15-54
Defining the Human Task Activity Title, Initiator, Priority, and Parameter Variables...... 15-55
Specifying the Task Title ............................................................................................. 15-55
Specifying the Task Initiator and Task Priority ........................................................... 15-55
Specifying Task Parameters ....................................................................................... 15-56
Defining the Human Task Activity Advanced Features ........................................... 15-57
Specifying a Scope Name and a Global Task Variable Name ..................................... 15-58
Specifying a Task Owner ............................................................................................ 15-58
Specifying an Identification Key ................................................................................ 15-59
Including the Task History of Other Human Tasks ................................................... 15-59
Allowing Task and Routing Customizations in BPEL Callbacks .................................. 15-59
Viewing the Generated Human Task Activity ......................................................... 15-60
BPEL Callbacks ........................................................................................................ 15-62
Including the Task History from Other Workflows ................................................. 15-64
Outcome-Based Modeling ......................................................................................... 15-64
Payload Updates ....................................................................................................... 15-65
Case Statements for Other Task Conclusions ......................................................... 15-65

Task 3: Generating the Task Display Form ............................................................... 15-66
Overview of Task Display Forms ................................................................................ 15-66
Selecting a Task Display Form ................................................................................... 15-66
Preview Release of Task Display Form Support for ADF Data Controls .................. 15-67
Automatically Generating a Simple Task Display Form ......................................... 15-68
Payload File for the Autogenerated JSP ................................................................... 15-68
Generating a Custom Task Display Form ................................................................. 15-74
Autogenerated JSP .................................................................................................... 15-75
Custom JSP ............................................................................................................... 15-76
Default JSP .............................................................................................................. 15-77
XSL ......................................................................................................................... 15-77
Deploying Task Display Forms .................................................................................. 15-77
Creating Custom JSP Forms ...................................................................................... 15-78
Adding Update Support in the Custom JSP .............................................................. 15-80
Adding Update Support in the Custom JSP .............................................................. 15-80

How Changes to a Workflow Appear in Worklist Application ................................. 15-80
Notifications from Workflow Services ....................................................................... 15-80
Configuring the Notification Channel ...................................................................... 15-81
Contents of Notification .......................................................................................... 15-82
Configuring Messages in Different Languages ....................................................... 15-83
Sending Actionable E-mails ..................................................................................... 15-83
Sending Inbound and Outbound Attachments ......................................................... 15-84
Sending Inbound Comments .................................................................................... 15-84
Reliability Support .................................................................................................. 15-84
Sending Secure Notifications .................................................................................. 15-85
Channels Used for Notifications ............................................................................. 15-85
Sending Reminders .................................................................................................. 15-85
End-to-End Workflow Examples ............................................................................... 15-86
Vacation Request Example ....................................................................................... 15-87
Prerequisites ............................................................................................................. 15-87
Modeling the Vacation Request Process ................................................................... 15-88
Creating the Vacation Request Process and Importing the Schema ....................... 15-89

xii
16 Worklist Application

Add a Human Task to the Order Approval Process.............................. 15-89
Assigning Input and Output Parameters for the Human Task........... 15-92
Modeling the Task Outcome.............................................................. 15-94
Validating, Compiling, and Deploying the Order Approval Process.... 15-95
Running the Order Approval Process.................................................. 15-95
Workflow Services................................................................................ 15-97
EJB, SOAP, and Java Support for the Workflow Services................ 15-97
Security Model for Services................................................................. 15-98
Security in SOAP Web Services......................................................... 15-98
Creating Workflow Context on Behalf of a User............................... 15-99
Task Service......................................................................................... 15-99
Task Query Service............................................................................ 15-101
Identity Service.................................................................................. 15-102
Creating Users and Groups................................................................. 15-104
Identity Service Providers................................................................. 15-104
User and Role Properties.................................................................. 15-106
Multirealm Support........................................................................... 15-107
Authentication, Authorization, and Identity Service Providers.......... 15-108
Notification Service.......................................................................... 15-108
Task Metadata Service........................................................................ 15-109
User Metadata Service....................................................................... 15-109
Runtime Config Service..................................................................... 15-111
Internationalization of Attribute Labels........................................... 15-113
Configuring the Assignment Service................................................. 15-113
Dynamic Assignment Functions....................................................... 15-113
Implementing a Dynamic Assignment Function............................. 15-114
Configuring Dynamic Assignment Functions.................................... 15-117
Configuring Display Names for Dynamic Assignment Functions...... 15-118
Dynamically Assigning Task Participants with the Assignment Service 15-118
Assignment Service Overview.......................................................... 15-119
Implementing an Assignment Service................................................ 15-119
Example of Assignment Service Implementation............................ 15-120
Deploying a Custom Assignment Service........................................... 15-121
Custom Escalation Function............................................................... 15-122
Workflow Service and Identity Service Related XPath Extension Functions 15-122
Deprecated Workflow Service and Identity Service Functions........ 15-123
NLS Configuration............................................................................. 15-124
Summary.......................................................................................... 15-124

16 Worklist Application

Use Cases for the Worklist Application.................................................. 16-1
Overview of Worklist Application Concepts........................................ 16-2
Worklist Application User Types......................................................... 16-3
Task Components................................................................................ 16-3
Features of the Worklist Application.................................................. 16-4
Part IV  Development and Deployment Life Cycle

19  BPEL Process Deployment and Domain Management

Compiling and Deploying a BPEL Process ................................................................. 19-1
Compiling and Deploying in Oracle JDeveloper ...................................................... 19-2
20 Testing BPEL Processes

Overview of the BPEL Test Framework ........................................................................... 20-1
Test Cases Overview ........................................................................................................ 20-2
Test Suites Overview ....................................................................................................... 20-2
Emulations Overview ...................................................................................................... 20-2
Assertions Overview ........................................................................................................ 20-2
Process Code Coverage Overview .................................................................................. 20-3
JUnit Support Overview ................................................................................................. 20-3

Components of a Test Suite........................................................................................... 20-3
Process Initiation ............................................................................................................ 20-4
Emulations ...................................................................................................................... 20-4
Assertions ....................................................................................................................... 20-5
Include Files .................................................................................................................. 20-6

Creating Test Suites in Oracle JDeveloper ................................................................. 20-6
Creating Test Suites in Oracle JDeveloper.................................................................... 20-6
## 21 Oracle BPEL Portlets

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>OracleAS Portal Introduction</td>
<td>21-1</td>
</tr>
<tr>
<td>Step 1: Installing and Configuring the Required Oracle Application Server Components</td>
<td>21-3</td>
</tr>
<tr>
<td>Configuring Realms (10.1.3.1.0 Only)</td>
<td>21-3</td>
</tr>
<tr>
<td>Step 2: Deploying the Portlets</td>
<td>21-4</td>
</tr>
<tr>
<td>Deploying Portlets with dcmctl</td>
<td>21-4</td>
</tr>
<tr>
<td>Deploying Portlets with Oracle Enterprise Manager 11g Application Server Control Console</td>
<td>21-4</td>
</tr>
<tr>
<td>Step 3: Registering Web Providers with OracleAS Portal</td>
<td>21-6</td>
</tr>
<tr>
<td>Step 4: Defining Portlet Parameters and Accessing Portlet Data Sources</td>
<td>21-9</td>
</tr>
<tr>
<td>Defining Oracle BPEL Control Report Portlet Parameters and Accessing Portlet Data Sources</td>
<td>21-9</td>
</tr>
<tr>
<td>Instance State</td>
<td>21-11</td>
</tr>
<tr>
<td>Instance Execution Time</td>
<td>21-11</td>
</tr>
<tr>
<td>Performance</td>
<td>21-12</td>
</tr>
<tr>
<td>Activity Sensor</td>
<td>21-13</td>
</tr>
<tr>
<td>Process Time Distribution</td>
<td>21-13</td>
</tr>
<tr>
<td>Fault Sensor</td>
<td>21-13</td>
</tr>
<tr>
<td>Defining Oracle BPEL Worklist Application Portlet Parameters and Accessing Portlet Data Sources</td>
<td>21-13</td>
</tr>
<tr>
<td>Listing Portlet Customization</td>
<td>21-14</td>
</tr>
<tr>
<td>Listing Portlet View</td>
<td>21-16</td>
</tr>
<tr>
<td>Analysis Portlet Customization</td>
<td>21-17</td>
</tr>
</tbody>
</table>
22 Oracle BPEL Control Reports

Creating Oracle BPEL Control Reports ................................................................. 22-1
  Creating Process Reports ......................................................................................... 22-3
  Creating Performance Reports .................................................................................. 22-5
  Creating Activity Sensor Reports ............................................................................ 22-7
  Creating Fault Sensor Reports .................................................................................. 22-9
  Creating Process Time Distribution Reports ............................................................ 22-10

Summary ......................................................................................................................... 22-11

Part V Reference Information

A Troubleshooting and Workarounds

Troubleshooting General Issues .................................................................................. A-1
  Setting Properties for BPEL Processes to Successfully Complete and Catch Exception A-1
  Developer Prompt on Windows 2000 ........................................................................ A-2
  Correcting Validation Errors in Complex Processes .................................................. A-2
  Handling Long-Running Processes .......................................................................... A-2
  Creating an Empty BPEL Process and Importing a Schema ...................................... A-3

Troubleshooting Sensors—The Custom Data Publisher ............................................... A-3
  Data Publisher Is Not Working ................................................................................ A-3
  Data Publisher Works, But Business Process Runs Slowly ......................................... A-4
  Caching Data in the Data Publisher Is Not Supported .............................................. A-4
  Unexpected Errors in the Data Publisher ................................................................ A-4
  Data Extracted to XML Is Difficult to Work With ................................................... A-4

Troubleshooting Oracle BPEL Worklist Application .................................................. A-5
  Not Able to Log In to the Worklist Application ......................................................... A-5
  Information Is Displayed in a Different Language .................................................... A-5
  Dates and Times Are Displayed Incorrectly ............................................................. A-5
  The User Is Not Permitted to Perform an Action ...................................................... A-5
  Expected Task Is Not Listed Under Task Titles ....................................................... A-5

Summary ......................................................................................................................... A-6

B BPEL Process Activities and Services

Process Activities Overview ....................................................................................... B-1
  Tabs Common to Many Activities .............................................................................. B-2
  Assign Activity ........................................................................................................ B-3
  Compensate Activity ............................................................................................... B-4
  Decide Activity ......................................................................................................... B-5
  Email Activity ......................................................................................................... B-6
  Empty Activity ........................................................................................................ B-6
  Fax Activity ............................................................................................................. B-9
Deployment Descriptor Properties

Deployment Descriptor Preference Properties

- Defining a Preference Property .................................................. C-1
- Updating a Preference at Run Time ........................................ C-1
- Getting the Value of a Preference within a BPEL Process ........... C-3
- Encrypting a Preference Value .............................................. C-3

Deployment Descriptor Configuration Properties

- Defining a Configuration Property ........................................... C-5

Summary .............................................................................. C-9
D XPath Extension Functions

Advanced Functions
create-nodeset-from-delimited-string .............................................. D-2
generate-guid ................................................................. D-2
lookup-dvm ............................................................... D-2
lookup-xnl ................................................................. D-3

BPEL Extension Functions
getLinkStatus ................................................................. D-4
getVariableData ............................................................. D-4
getVariableProperty .......................................................... D-5

BPEL XPath Extension Functions
addQuotes ................................................................. D-5
appendToList ............................................................ D-6
copyList ................................................................. D-6
countNodes ............................................................... D-7
doc ................................................................. D-7
formatDate .............................................................. D-8
generateGUID .......................................................... D-8
getContentAsString ..................................................... D-8
getConversationId .......................................................... D-9
getCreator ................................................................. D-9
generateGUID .......................................................... D-9
getProcessURL ........................................................... D-9
generateGUID .......................................................... D-9
getProcessId ............................................................. D-10
gerupdateTime .......................................................... D-10
gerupdatedId ........................................................... D-10
gerupdatedElement ....................................................... D-10
gerupdatedGroupIdFromGroupAlias ...................................... D-11
gerupdatedInstantId ..................................................... D-11
gerupdatedNodeValue .................................................... D-11
gerupdatedNodes ........................................................ D-11
gerupdatedPreference ..................................................... D-12
gerupdatedProcessId ....................................................... D-12
gerupdatedProcessOwnerld ............................................... D-12
gerupdatedProcessURL ................................................ D-12
gerupdatedProcessVersion .............................................. D-13
gerupdatedUserAliasId ................................................... D-13

integer ................................................................. D-13
parseEscapedXML ....................................................... D-13
processXQuery ........................................................ D-14
processXSL ............................................................... D-14
processXSL1 .............................................................. D-14
readBinaryFromField .................................................. D-15
readFile ................................................................. D-15
writeBinaryToField ..................................................... D-15

lookup-table .......................................................... D-16
query-database .......................................................... D-17
sequence-next-val ...................................................... D-17

Date Functions .................................................. D-18
add-dayTimeDuration-to-dateTime .................... D-18
current-date .................................................. D-18
current-dateTime ........................................... D-19
current-time ............................................... D-19
day-from-dateTime ....................................... D-19
format-dateTime ........................................... D-20
hours-from-dateTime .................................. D-20
implicit-timezone ........................................ D-20
minutes-from-dateTime ............................... D-21
month-from-dateTime ................................ D-21
seconds-from-dateTime ............................... D-21
subtract-dayTimeDuration-from-dateTime .......... D-21
timezone-from-dateTime ............................... D-22
year-from-dateTime ................................... D-22

Mathematical Functions ........................................ D-23
abs ........................................................................ D-23

Identity Service Functions ................................... D-23
defaultRealmName ........................................ D-23
groupProperty ............................................ D-24
getManager ............................................... D-24
getReportees ............................................... D-25
getSupportedRealmNames .......................... D-25
getUserProperty ......................................... D-25
getUserRoles ............................................... D-26
getUserInGroup ........................................ D-26
getUserInRole ........................................... D-27
lookupGroup ............................................. D-28
lookupUser ............................................... D-28

Workflow Service Functions ................................. D-28
clearTaskAssignees .................................... D-28
createWordMLDocument .............................. D-29
getNotificationProperty ......................... D-29
getNumberOfTaskApprovals .................... D-30
getPreviousTaskApprover ....................... D-30
getTaskAttachmentByName ....................... D-30
getTaskAttachmentByIndex ..................... D-30
getTaskAttachmentContents .................... D-31
getTaskAttachmentsCount ...................... D-31
getResourceBundleString ....................... D-31
wfDynamicGroupAssign ......................... D-32
wfDynamicUserAssign ......................... D-33

String Functions ............................................... D-33
compare .................................................. D-34
compare-ignore-case .................................. D-34
create-delimited-string ............................. D-35
E  Workflow Services Changes Between 10.1.2 and 10.1.3.1

Backwards Compatibility between 10.1.2 and 10.1.3.1 ......................................................... E-1
Changes Between the Workflow Wizard and the Human Task Editor ....................................... E-2
Changes to Configuration Files .............................................................................................. E-6
Changes to Worklist APIs ........................................................................................................ E-7
Changes to Task Display Form Deployment ............................................................................. E-8
Changes to the Oracle BPEL Worklist Application .................................................................... E-10
Changes to Oracle BPEL Control ............................................................................................. E-10
Migrating Workflow Definitions from 10.1.2 to 10.1.3.1 ............................................................ E-11

Index
This manual describes how to use Oracle BPEL Process Manager.

This preface contains the following topics:

- Audience
- Documentation Accessibility
- Related Documents
- Conventions

**Audience**

This manual is intended for anyone who is interested in using Oracle BPEL Process Manager.

**Documentation Accessibility**

Our goal is to make Oracle products, services, and supporting documentation accessible, with good usability, to the disabled community. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Accessibility standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For more information, visit the Oracle Accessibility Program Web site at http://www.oracle.com/accessibility/

**Accessibility of Code Examples in Documentation**

Screen readers may not always correctly read the code examples in this document. The conventions for writing code require that closing braces appear on an otherwise empty line; however, some screen readers may not always read a line of text that consists solely of a bracket or brace.

**Accessibility of Links to External Web Sites in Documentation**

This documentation may contain links to Web sites of other companies or organizations that Oracle does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.
TTY Access to Oracle Support Services

Oracle provides dedicated Text Telephone (TTY) access to Oracle Support Services within the United States of America 24 hours a day, seven days a week. For TTY support, call 800.446.2398.

Related Documents

For more information, see the following Oracle resources:

- Oracle BPEL Process Manager Quick Start Guide
- Oracle BPEL Process Manager Order Booking Tutorial
- Oracle BPEL Process Manager Administrator’s Guide
- Oracle Adapters for Files, FTP, Databases, and Enterprise Messaging User’s Guide
- Oracle Application Server Adapter Concepts
- Oracle Application Server Adapter for Oracle Applications User’s Guide

Printed documentation is available for sale in the Oracle Store at http://oraclestore.oracle.com/

To download free release notes, installation documentation, white papers, or other collateral, visit the Oracle Technology Network (OTN). You must register online before using OTN; registration is free and can be done at http://www.oracle.com/technology/membership/

To download Oracle BPEL Process Manager documentation, technical notes, or other collateral, visit the Oracle BPEL Process Manager site at Oracle Technology Network (OTN): http://www.oracle.com/technology/bpel/

If you already have a username and password for OTN, then you can go directly to the documentation section of the OTN Web site at http://www.oracle.com/technology/documentation/

See the Business Process Execution Language for Web Services Specification, available at the following URL:


See the XML Path Language (XPath) Specification, available at the following URL:

http://www.w3.org/TR/1999/REC-xpath-19991116

See the Web Services Description Language (WSDL) 1.1 Specification, available at the following URL:

http://www.w3.org/TR/wsd1

Conventions

The following text conventions are used in this document:
<table>
<thead>
<tr>
<th>Convention</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>boldface</strong></td>
<td>Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.</td>
</tr>
<tr>
<td><em>italic</em></td>
<td>Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.</td>
</tr>
<tr>
<td>monospace</td>
<td>Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.</td>
</tr>
</tbody>
</table>
The new features of Oracle BPEL Process Manager 10g (10.1.3.1.0) include:

- Redesign of human task workflow, which includes
  - A new design-time interface: a re-entrant Human Task editor for declarative task configuration
  - Metadata-driven workflow with minimal BPEL code generated
  - A modeling tool for creating and configuring complex patterns
  - Built-in dispatching functions: round-robin, least-busy, and most-productive
  - Dynamic assignment APIs for writing custom assignment services
  - New demos: OrderApproval, HelpDeskRequest, and ExpenseApproval

See Chapter 15, "Oracle BPEL Process Manager Workflow Services" for more information.

- New Worklist Application functionality, which includes
  - Support for user profiles: end user, supervisor, process owner, group owner, and administrator
  - Support for custom work queues and proxy support
  - Ability to define custom vacation rules and delegation rules
  - Admin pages for managing rules and flex field mappings
  - An improved design that supports enhancements to search capabilities, identity browser, user and group rules, page customization, and more

See Chapter 16, "Worklist Application" for more information.

- A comprehensive unit testing framework for BPEL, which enables you to
  - Automate testing of BPEL processes
  - Emulate partners and services and specify your own return data (instead of actually invoking those services)
  - Create assertions to verify that your process works as expected
  - Calculate code coverage and highlight code that was not run
– Create unit tests from a BPEL audit trail
– Integrate into Ant-JUnit reports
See Chapter 20, "Testing BPEL Processes" for more information.

■ Integration of Oracle BPEL Process Manager with business rules and the decision service
– Design-time support for business rules engines with a new decision service activity and wizard
– Design-time integration for Oracle Business Rules and iLog JRules
See Chapter 18, "BPEL Process Integration with Business Rules" for more information.

■ Integration of Oracle BPEL Process Manager with JAAS and application server J2EE security
See the following for more information:
– "Logging into Domains" on page 199
– "Oracle BPEL Control and Oracle BPEL Admin Console Users and Roles" of Oracle BPEL Process Manager Administrator's Guide

■ Adapter improvements, which include
– File/FTP adapter: using an invoke activity, you can read a file synchronously using the File adapter or get a file using the FTP adapter
– MQSeries adapter: a new adapter that exposes the JMS functionality provided by IBM WebSphere MQ and native WebSphere MQ functionality
See the following for more information:
– Oracle Adapters for Files, FTP, Databases, and Enterprise Messaging User’s Guide for more information on all the adapters.
– The online help in the Adapter Configuration Wizard

■ Oracle JDeveloper enhancements, which enable you to
– Create a custom template from an existing BPEL process
– Generate WSDLs with Java and EJB WSIF bindings automatically
– Import a schema during project creation
– Bookmark specific activities in a BPEL process to locate them quickly
– Search for and jump to a specific activity in a BPEL process, show and hide types of activities, and zoom in on containers (helpful for large, complex processes)
– Optimize a BPEL diagram layout
See the online help in Oracle JDeveloper for more information.

■ bpelx extensions for XML data manipulation
See "Manipulating XML Data with bpelx Extensions" on page 3-10 and the online help in Oracle JDeveloper for more information.

■ Improved clustering support: You can deploy to one node in a cluster with automatic deployment to all other nodes, because deployment suitcases are now stored in the dehydration store.
See Oracle BPEL Process Manager Installation Guide for more information.

- Support for standard ant.

- Deployment to multiple environments with different configuration values using the customize ant task

  The customize ant task enables you to specify the property values for development and production environments in a single build file location. The customize task captures the changes between different versions of bpel.xml using the build file.

  See "Deploying to Multiple Environments with Different Configuration Values" on page 19-5 for more information.
Part I

Introduction and Concepts

This part introduces Oracle BPEL Process Manager.
This part contains the following chapters:

- Chapter 1, "Introduction to Oracle BPEL Process Manager"
- Chapter 2, "Getting Started with Oracle BPEL Process Manager"
Introduction to Oracle BPEL Process Manager

This chapter provides a brief introduction to the Business Process Execution Language (BPEL), Oracle BPEL Process Manager, and Oracle JDeveloper, which enables you to design BPEL processes. An overview of how to use the information in this guide and references to additional tutorials and demonstrations installed with Oracle BPEL Process Manager are also provided.

This chapter contains the following topics:

- What Is BPEL?
- What Is Oracle BPEL Process Manager?
- What Is Oracle JDeveloper?
- How to Use This Guide
- Getting Started with Demonstrations, Activity and Conceptual References, and Tutorials
- Summary

**Note:** Oracle recommends that you perform the tutorials described in *Oracle BPEL Process Manager Quick Start Guide* and *Oracle BPEL Process Manager Order Booking Tutorial* before using this guide. These tutorials provide you with an introduction to designing and deploying BPEL processes.

What Is BPEL?

BPEL is an XML-based language for enabling task sharing across multiple enterprises using a combination of Web services. BPEL is based on the XML schema, simple object access protocol (SOAP), and Web services description language (WSDL). BPEL provides enterprises with an industry standard for business process orchestration and execution. Using BPEL, you design a business process that integrates a series of discrete services into an end-to-end process flow. This integration reduces process cost and complexity. The BPEL language enables you to define how to:

- Send XML messages to, and asynchronously receive XML messages from, remote services
- Manipulate XML data structures
- Manage events and exceptions
What Is Oracle BPEL Process Manager?

Oracle BPEL Process Manager provides a framework for easily designing, deploying, monitoring, and administering processes based on BPEL standards. Oracle BPEL Process Manager provides support for the following features:

- Web service standards such as XML, SOAP, and WSDL
- Dehydration (enables the states of long-running processes to be automatically maintained in a database) and correlation of asynchronous messages
- Service-oriented architecture (SOA)
- Parallel processing of tasks
- Fault handling and exception management during both design time and run time
- Event timeouts and notifications
- Compensation mechanisms for the implementation of long-running transactions
- Scalability and reliability of processes
- Management and administration of processes
- Version control
- Audit trails for tracing business flow history
- Installation on multiple operating systems and integration with multiple application servers (for example, Oracle Application Server, BEA WebLogic, and JBoss) and databases.

Oracle BPEL Process Manager adds value and ease of use to BPEL functionality by providing support for the following in Oracle JDeveloper:

- Transformations, workflows, worklists, notifications, sensors, and business rules
- Technology adapters (file, FTP, database, advanced queuing (AQ), Java Messaging Service (JMS), IBM WebSphere MQ, and Oracle Applications for Oracle E-Business Suite)
- Third-party adapters, including J.D. Edwards OneWorld, PeopleSoft, SAP R/3, Siebel, Tuxedo, CICS, VSAM, IMS/TM, and IMS/DB

Oracle BPEL Process Manager can also be integrated with Oracle Business Activity Monitoring, Oracle Application Server Portal, Oracle Application Server Integration B2B, and Oracle Application Server Integration InterConnect.

See Also:

- [http://www.oracle.com/technology/bpel](http://www.oracle.com/technology/bpel) for specific BPEL details, including links to BPEL specifications, white papers, product demonstrations, and discussion groups
- Chapter 3, "Manipulating XML Data in BPEL" through Chapter 12, "Interaction Patterns" for a review of key BPEL development concepts and code samples
What Is Oracle JDeveloper?

Oracle BPEL Process Manager provides support for using Oracle JDeveloper to graphically design BPEL processes.

Oracle JDeveloper is an integrated development environment (IDE) for building applications and Web services using Java, XML, and SQL standards. Oracle JDeveloper supports the entire development life cycle with integrated features for designing, coding, debugging, testing, profiling, tuning, and deploying applications. A visual and declarative development approach and the Oracle Application Development Framework (ADF) work together to simplify application development and reduce coding tasks.

Oracle JDeveloper uses BPEL as its native format. This means that processes built with Oracle JDeveloper are 100% portable. Oracle JDeveloper also enables you to view and modify the BPEL source without decreasing the usefulness of the tool.

You design BPEL processes by dragging and dropping elements (known as activities) into the process and editing their property pages. This eliminates the need to write BPEL code. You integrate BPEL processes with external services (known as partner links). You also integrate adapters and services such as workflows, transformations, notifications, sensors, worklist task management, and business rules with the process. Oracle JDeveloper can deploy the developed processes directly to Oracle BPEL Server. This facilitates the development and maintenance of BPEL processes.

See Also:

- "Sensor Integration with Oracle Business Activity Monitoring" on page 17-13 for details about integrating Oracle BPEL Process Manager with Oracle Business Activity Monitoring
- Chapter 21, "Oracle BPEL Portlets" for details about integrating Oracle BPEL Process Manager with OracleAS Portal
- Oracle Application Server Integration InterConnect User's Guide for details about integrating Oracle BPEL Process Manager with Oracle Application Server Integration InterConnect
- Oracle BPEL Process Manager Quick Start Guide for additional Oracle BPEL Process Manager introductory details
- Oracle Adapters for Files, FTP, Databases, and Enterprise Messaging User's Guide for details about supported technology adapters
- Oracle Application Server Adapter for Oracle Applications User's Guide
- Oracle Application Server Adapter Concepts
- Oracle BPEL Process Manager Installation Guide for a list of supported operation systems
- The following URL for additional details about Oracle BPEL Process Manager support for third-party adapters:

Introduction to Oracle BPEL Process Manager 1-3
Oracle BPEL Process Manager provides support for the following services and adapters in Oracle JDeveloper:

- Transformations, workflows, worklists, notifications, sensors, and business rules
- Technology adapters (file, FTP, database, AQ, JMS, MQ, and Oracle Applications)

Figure 1–1 shows Oracle JDeveloper with a BPEL process being designed.

See Also:
- "Overview of BPEL Project Creation and Oracle JDeveloper" on page 2-3 for a description of the sections of Oracle JDeveloper shown in Figure 1–1
- Oracle BPEL Process Manager Quick Start Guide and Oracle BPEL Process Manager Order Booking Tutorial for tutorials that use Oracle JDeveloper
- Online Help available from the Help main menu for additional details about Oracle JDeveloper

How to Use This Guide

This guide is divided into several parts designed to first familiarize you with key BPEL development concepts and features and then describe how Oracle BPEL Process Manager adds value and ease of use to BPEL functionality. This guide layout is described in Table 1–1.
### Table 1–1  Developer’s Guide Contents

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
</table>
| **Part I, “Introduction and Concepts”** | Chapters in this part provide an overview of the following topics:  
- BPEL specifications, Oracle BPEL Process Manager, and Oracle JDeveloper  
- Demonstrations, tutorials, and activity and conceptual references provided with Oracle BPEL Process Manager  
- Starting and stopping key Oracle BPEL Process Manager components  
- An introduction to Oracle JDeveloper, including an overview of designer window sections, and a description of project files and the drag-and-drop activity functionality you follow to design and deploy a BPEL process  
- Oracle BPEL Control for running deployed BPEL processes |
| **Part II, “Reviewing Key BPEL Development Concepts and Code Samples”** | Chapters in this part introduce you to key BPEL development concepts and associated code samples. These chapters are useful for any developer interested in understanding the underlying functionality of BPEL. Specific topics discussed include the following:  
- XML document manipulation  
- Synchronous and asynchronous services invocation  
- Parallel flows  
- Conditioning branching  
- Fault handling and exception management  
- Java/J2EE code integration in BPEL processes  
- Events and timeouts  
- BPEL process invocation  
- Interaction patterns |
| **Part III, “Oracle BPEL Process Manager Services”** | Once you have gained a solid knowledge of the key BPEL development concepts described in Part II, you are ready to learn how Oracle BPEL Process Manager adds value and ease of use to BPEL functionality to provide support for the following services:  
- Transformations  
- Notifications  
- Workflows  
- Worklists  
- Sensors  
- Business rules |
| **Part IV, “Development and Deployment Life Cycle”** | Chapters in this part describe how to run and manage deployed BPEL processes from Oracle BPEL Control, how to test BPEL processes in a preproduction environment, how to create runtime reports, and how to integrate with OracleAS Portal. |
| **Part V, “Reference Information”** | Appendices in this part provide reference details about troubleshooting, supported activities, deployment descriptor properties, and XPath expression functions. |
Getting Started with Demonstrations, Activity and Conceptual References, and Tutorials

In addition to the contents of this guide, the Oracle BPEL Process Manager Quick Start Guide, and the Oracle BPEL Process Manager Order Booking Tutorial, a series of demonstrations, activity and conceptual reference materials, and tutorials are also provided to increase conceptual knowledge and hands-on experience with Oracle BPEL Process Manager. These materials are installed with Oracle BPEL Process Manager in the SOA_Oracle_Home\bpel\samples directory. Review the README.txt file in the samples directory for instructions on using these samples.

Table 1–2 describes the contents of the samples directory. If you are using Oracle JDeveloper, you can also access details about this directory from the Start Menu by selecting Start > All Programs > Oracle - Oracle_Home > Oracle BPEL Process Manager > Getting Started with Samples.

You can automatically create the BPEL project for a sample by performing the following steps:

1. Select an application in the Application Navigator.
2. Select Open from the File main menu in Oracle JDeveloper.
3. Go to the directory of the sample you want to use.
4. Select the .jpr file of the sample.

This causes the BPEL project for the selected sample to display in the Application Navigator.

New samples are periodically added. Visit the Oracle BPEL Process Manager site on the Oracle Technology Network (OTN) periodically for information about downloading any new samples:


---

**Table 1–2** Tutorials, Demonstrations, and Reference Materials

<table>
<thead>
<tr>
<th>Directory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>demos</td>
<td>Contains a set of common business scenarios and describes how they are implemented with BPEL. Table 1–3 on page 1–7 provides descriptions of the available demonstrations.</td>
</tr>
</tbody>
</table>
Getting Started with Demonstrations, Activity and Conceptual References, and Tutorials

Introduction to Oracle BPEL Process Manager

Table 1–3 demos Directory Contents

<table>
<thead>
<tr>
<th>Directory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AmazonFlow</td>
<td>Describes how to integrate an Amazon Web service with a BPEL process to search for an item.</td>
</tr>
<tr>
<td>Attachment</td>
<td>Describes how to use binary file attachments in SOAP messages with the Direct Internet Message Encapsulation (DIME) and Multipurpose Internet Message Extensions (MIME) protocols.</td>
</tr>
</tbody>
</table>
| AutoLoanDemo | Describes how to integrate Oracle BPEL Process Manager with a backend business rules engine. A BPEL process is modeled that uses the decision service to perform the following:  
  - Calculate a credit rating for a customer loan request  
  - Provide a recommendation on the bank and APR for the requested loan  
  The output of the decision service is passed to a human task for modification before the loan request is approved or rejected. |
| BankTransferDemo | Describes how to perform a bank transfer. This sample illustrates the ability of Oracle BPEL Process Manager transaction management. The sample shows two types of transaction management:  
  - Internal engine-implemented JTA transaction management  
  - Explicit compensating transactions modeled in BPEL |
| BPELTest | Describes several of the BPEL test framework features. The test framework provides a structured way to test BPEL processes and alleviates common problems like dependencies on complex external systems and performing data assertions. |
| CheckoutDemo | Describes an interaction between a Java Server Page (JSP) user interface and a BPEL process |

What Demonstrations Are Available?

Table 1–3 describes the BPEL process demonstrations available for use in the demos directory. See the documentation available in these directories for instructions on running these demonstrations.

Table 1–3 (Cont.) Tutorials, Demonstrations, and Reference Materials

<table>
<thead>
<tr>
<th>Directory</th>
<th>Description</th>
</tr>
</thead>
</table>
| interop | Contains a set of BPEL projects showing the interoperability of Oracle BPEL Process Manager with Web services implemented with the following:  
  - Microsoft Net  
  - Apache Axis  
  - BEA WebLogic |
| references | Contains activities and concepts defined in the BPEL language. Table 1–4 on page 1-9 provides descriptions of the available activities and concepts. |
| tutorials | Contains a set of BPEL processes targeting the various BPEL tasks to which you are exposed. Table 1–5 on page 1-10 provides descriptions of the available tutorials. |
| utils | Contains a set of building block services shared by the BPEL samples |

Introduction to Oracle BPEL Process Manager 1-7
### Table 1-3 (Cont.) demos Directory Contents

<table>
<thead>
<tr>
<th>Directory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DocumentReview</td>
<td>Describes how to create a business process for reviewing a document in parallel. A final reviewer reviews comments from each of the group reviewers. A workflow application views and acts on the tasks. This example highlights the use of the following features: ■ Modeling a group vote participant type in the Oracle JDeveloper environment ■ Using Oracle BPEL Worklist Application to view and act on tasks</td>
</tr>
<tr>
<td>ExpenseRequestApproval</td>
<td>Describes how to approve or reject an expense request from an employee. This demonstrates management chain approval and use of the decision service to determine the levels of approval required for a particular expense request.</td>
</tr>
<tr>
<td>GoogleFlow</td>
<td>Describes how to invoke a Google Web service from a BPEL process</td>
</tr>
<tr>
<td>HelpDeskServiceRequest</td>
<td>Describes how to process a help desk service request. The demonstration uses an ad hoc participant type for accepting or rejecting a service request.</td>
</tr>
<tr>
<td>HotwireDemo</td>
<td>This sample illustrates the asynchronous multistep conversation between two BPEL processes. One BPEL process initiates the conversation and sends the message to the other BPEL process. The second process waits for 30 seconds and responds asynchronously. Then the first process waits for 30 seconds and calls the second process again. The second process responds again after 30 seconds.</td>
</tr>
<tr>
<td>IBM Samples</td>
<td>Describes how to execute the BPEL samples shipping with the IBM Business Process Execution Language for Web Services Java Run Time (BPWS4J) on Oracle BPEL Server</td>
</tr>
<tr>
<td>LoanDemo</td>
<td>Describes how to integrate a synchronous credit rating service and two asynchronous loan processor services into an end-to-end loan procurement application with a Java Server Page (JSP) user interface to initiate the process and view loan offer results</td>
</tr>
<tr>
<td>LoanDemoPlus</td>
<td>Describes how to extend the LoanDemo sample to use Java embedding exception management, including manual processing steps and development of a richer custom user interface</td>
</tr>
<tr>
<td>ParallelSearch</td>
<td>Describes how to use Oracle BPEL Server to perform parallel synchronous invocations. This sample illustrates how to use the nonBlockingInvoke property in bpel.xsd. This property enables you to execute a synchronous BPEL process calling multiple synchronous Web services in flow in real parallel mode. If you set the nonBlockingInvoke property to false, Oracle BPEL Server blocks the Web service call until the other is finished.</td>
</tr>
<tr>
<td>ResilientDemo</td>
<td>Describes how to use a BPEL process to manage fault handling and run-time exceptions</td>
</tr>
<tr>
<td>SalesforceFlow</td>
<td>Describes how to integrate the Salesforce.com sForce Web services into a BPEL process (including authentication, session management, and dynamic load balancing)</td>
</tr>
<tr>
<td>SleepBroker</td>
<td>Describes how to use a process that receives a number, creates that number of branches using the flowN activity, and waits for a period of time based on the index variable setting. This process receives an integer as input. It creates that number of branches using bpelx:flowN. In each branch, a wait activity is executed. The wait time is based on the index variable.</td>
</tr>
</tbody>
</table>
VacationRequest Describes how to approve or reject a vacation request. The approval or rejection is a one-step process involving the manager of the user filing the vacation request. This demonstration also describes the use of workflow for simple approvals, and the use of a deployment descriptor preference to replace a static parameter value in the BPEL process.

XMLMapper Describes how to create a transformation that maps a purchase order schema to an invoice schema.

What Activity and Conceptual References Are Available?

Table 1–4 describes the activity and conceptual references available for review and use in the references directory. The comment lines in each bpel.xml file and .bpel file describe the specific context in which the activity is being used.

<table>
<thead>
<tr>
<th>Directory</th>
<th>Activity Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assign</td>
<td>Shows how to receive an input string, prefix Hello to it using an assign activity, and asynchronously return the result</td>
</tr>
<tr>
<td>BPELTest</td>
<td>Illustrates the features of the BPEL test framework. The BPEL test framework provides emulation and assertion capabilities and eases the automation of testing BPEL processes.</td>
</tr>
<tr>
<td>Catch</td>
<td>Shows how an exception can be raised using the throw activity and managed using a catch activity</td>
</tr>
<tr>
<td>CustomXPathFunction</td>
<td>Shows how to use custom XPath functions within assign activities</td>
</tr>
<tr>
<td>DynamicPartnerLink</td>
<td>Shows how to update dynamic partner links</td>
</tr>
<tr>
<td>Event</td>
<td>Shows how to enable an asynchronous BPEL process and use event handlers to receive and process events while waiting for the asynchronous callback</td>
</tr>
<tr>
<td>Flow</td>
<td>Shows how to create parallel paths of execution within a BPEL process</td>
</tr>
<tr>
<td>FlowN</td>
<td>Shows how to receive an integer and create that number of branches</td>
</tr>
<tr>
<td>Invoke</td>
<td>Shows how to invoke a synchronous integer increment service</td>
</tr>
<tr>
<td>JavaExec</td>
<td>Shows how to use the BPEL exec extension to invoke a Java class from within a BPEL process</td>
</tr>
<tr>
<td>Link</td>
<td>Shows how a link defines dependencies between executions of activities. In this sample, a link in a flow activity sequences the execution of two service invocations.</td>
</tr>
<tr>
<td>Pick</td>
<td>Shows how to invoke an asynchronous loan service and use a BPEL pick activity to receive an asynchronous response or a timeout message. If the loan amount is more than 10000, it takes about 30 seconds for the server to process it, causing a timeout to be raised.</td>
</tr>
<tr>
<td>Receive</td>
<td>Shows how to invoke an asynchronous loan service and wait for an asynchronous callback message using the BPEL receive activity</td>
</tr>
<tr>
<td>Replay</td>
<td>Shows how to replay an activity, such as a scope</td>
</tr>
<tr>
<td>Reply</td>
<td>Shows how to receive a string as input, perform an assign, and use the reply activity to synchronously return the modified string</td>
</tr>
</tbody>
</table>

Table 1–4 (Cont.) demos Directory Contents

<table>
<thead>
<tr>
<th>Directory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assign</td>
<td>Shows how to receive an input string, prefix Hello to it using an assign activity, and asynchronously return the result</td>
</tr>
<tr>
<td>BPELTest</td>
<td>Illustrates the features of the BPEL test framework. The BPEL test framework provides emulation and assertion capabilities and eases the automation of testing BPEL processes.</td>
</tr>
<tr>
<td>Catch</td>
<td>Shows how an exception can be raised using the throw activity and managed using a catch activity</td>
</tr>
<tr>
<td>CustomXPathFunction</td>
<td>Shows how to use custom XPath functions within assign activities</td>
</tr>
<tr>
<td>DynamicPartnerLink</td>
<td>Shows how to update dynamic partner links</td>
</tr>
<tr>
<td>Event</td>
<td>Shows how to enable an asynchronous BPEL process and use event handlers to receive and process events while waiting for the asynchronous callback</td>
</tr>
<tr>
<td>Flow</td>
<td>Shows how to create parallel paths of execution within a BPEL process</td>
</tr>
<tr>
<td>FlowN</td>
<td>Shows how to receive an integer and create that number of branches</td>
</tr>
<tr>
<td>Invoke</td>
<td>Shows how to invoke a synchronous integer increment service</td>
</tr>
<tr>
<td>JavaExec</td>
<td>Shows how to use the BPEL exec extension to invoke a Java class from within a BPEL process</td>
</tr>
<tr>
<td>Link</td>
<td>Shows how a link defines dependencies between executions of activities. In this sample, a link in a flow activity sequences the execution of two service invocations.</td>
</tr>
<tr>
<td>Pick</td>
<td>Shows how to invoke an asynchronous loan service and use a BPEL pick activity to receive an asynchronous response or a timeout message. If the loan amount is more than 10000, it takes about 30 seconds for the server to process it, causing a timeout to be raised.</td>
</tr>
<tr>
<td>Receive</td>
<td>Shows how to invoke an asynchronous loan service and wait for an asynchronous callback message using the BPEL receive activity</td>
</tr>
<tr>
<td>Replay</td>
<td>Shows how to replay an activity, such as a scope</td>
</tr>
<tr>
<td>Reply</td>
<td>Shows how to receive a string as input, perform an assign, and use the reply activity to synchronously return the modified string</td>
</tr>
</tbody>
</table>
What Tutorials Are Available?

Table 1–5 describes the tutorials available for use in the tutorials directory. See the documentation available in these directories for instructions on running these tutorials.

<table>
<thead>
<tr>
<th>Directory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Switch</td>
<td>Shows how to use a switch activity to return a different text message based on whether the input value is greater or less than zero.</td>
</tr>
<tr>
<td>Terminate</td>
<td>Shows how to invoke a synchronous stock quoting service. The terminate activity then aborts, causing the final callback invoke activity to be skipped.</td>
</tr>
<tr>
<td>Throw</td>
<td>Shows how to throw a BPEL fault (without handling it) and cause the instance to fault.</td>
</tr>
<tr>
<td>Wait</td>
<td>Shows how to receive input, wait for 60 seconds, and asynchronously call back a client.</td>
</tr>
<tr>
<td>While</td>
<td>Shows how to invoke an incremental service n times with a while activity, where n is provided as an input value.</td>
</tr>
<tr>
<td>XPath</td>
<td>Shows how to receive an invalid application, perform several XPath copies, and asynchronously return the application. This showcases the use of namespace-qualified XPath query strings in assign activities.</td>
</tr>
<tr>
<td>XPathFunction</td>
<td>Shows how to define and use custom XPath functions within BPEL assign activities.</td>
</tr>
</tbody>
</table>

See Also:
- Chapter 3, "Manipulating XML Data in BPEL" through Chapter 12, "Interaction Patterns" for activity development concepts and code samples.
- Appendix B, "BPEL Process Activities and Services" for specific details about activities that you drag and drop in Oracle JDeveloper.

### Table 1–5 tutorials Directory Contents

<table>
<thead>
<tr>
<th>Directory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>101.HelloWorld</td>
<td>This sample takes a string as input, appends Hello to it, and asynchronously generates a greeting as a response.</td>
</tr>
<tr>
<td>102.InvokingProcesses</td>
<td>This sample invokes a variety of processes, including JSPs and remote method invocations (RMIs).</td>
</tr>
<tr>
<td>103.XMLDocuments</td>
<td>This sample shows how to use XML variables and the assign activity to manipulate XML documents.</td>
</tr>
<tr>
<td>104.SyncQuoteConsumer</td>
<td>This sample shows how to use the invoke activity to invoke a synchronous stock quote service.</td>
</tr>
<tr>
<td>105.AsyncCompositeLoanBroker</td>
<td>This sample shows how to use the receive activity to receive a callback from an asynchronous loan processor Web service.</td>
</tr>
<tr>
<td>106.ParallelFlows</td>
<td>This sample shows how to use the flow activity to define parallel paths of execution within a process. In this sample, two asynchronous loan processing services are invoked in parallel.</td>
</tr>
</tbody>
</table>
107. Exceptions
This sample shows how to use fault handling to manage faults generated by invoke and throw activities. The process uses a pick activity to receive the response from a loan validator. If an exception message is received, it throws an error that is handled in a catch fault handler.

108. Timeouts
This sample shows how to define and manage timeouts using the pick activity.

109. Correlation Sets
This sample shows how to use correlation sets to correlate asynchronous message exchanges between buyer and seller services. It shows content-based correlation of asynchronous messages.

112. Arrays
This sample shows how to design a BPEL process that uses arrays. This sample illustrates how you can handle array structures present in your XML payload by using the while activity in the BPEL process.

113. ARCADRouting
This sample shows how to coordinate the flow of messages across three services: A, B, and C.

114. XSLT Transformations
This sample shows how to invoke XSLT transformations to perform complex data manipulations. The process takes in complex invoice data as input. It uses the ora:processXSLT function to pass this data as input to be the XSLT service and returns the transformed content.

115. XQuery Transformations
This sample shows the use of XQuery functions in Oracle BPEL Process Manager. This sample requires XQuery libraries available only in the Oracle BPEL Process Manager for OracleAS Middle Tier installation type or one of the Oracle Application Server SOA installation types. This sample cannot be used with the Oracle BPEL Process Manager for Developers installation type.

121. FileAdapter
These samples show how to use the file adapter. The following tutorials are provided:

- COBOL Copybook — Processes native data defined using a COBOL copybook
- Complex structures — Processes native data defined in a custom format
- Debatching — Processes native data containing multiple messages defined in a custom format
- Flat structure — Processes address book entries from a CSV (Comma Separated Values) file. This is then transformed to a new address format (fixed-length format).
- Opaque with headers — Handles native data in an opaque format (for example, GIF or JPEG files)

See Also: Oracle Adapters for Files, FTP, Databases, and Enterprise Messaging User’s Guide
The following samples show how to use the database adapter:

- **advanced**—Advanced samples that insert and extract XML data stored in a database as a CLOB, set up sequencing, and insert into multiple databases as part of a single transaction.
- **Delete**—A record is passed to the operation and the database row with the primary key is deleted.
- **File2StoredProcedure**—Data is provided to a stored procedure, which is then executed.
- **File2Table**—The file adapter, XSLT Mapper, and database adapter take an inbound purchase order, transform it to another order format, and produce an outbound message.
- **Insert**—A record is passed to the operation and inserted into the database as relational data.
- **InsertWithCatch**—Adds fault handling to an insert operation.
- **MasterDetail**—Replicates data in the table of one database to the tables of another database.
- **Merge**—A record is passed to the operation and a database row is either inserted or updated.
- **PollingControlTableStrategy**—An inbound operation polls XML instances. A control table stores the primary key of every row that has yet to be processed.
- **PollingLastReadIdStrategy**—An inbound operation polls XML instances. A helper table remembers the last-updated value. A sequence value of 1000 means that every record with a sequence less than that value has already been processed.
- **PollingLastUpdatedStrategy**—An inbound operation polls XML instances. A helper table remembers the last-updated value.
- **PollingLogicalDeleteStrategy**—An inbound operation polls XML instances. A special field is updated on each row processed. The WHERE clause is updated at run time to filter out processed rows.
- **QueryByExample**—An outbound query by example operation.
- **RefCursor**—A BPEL process takes user input and executes a stored procedure. Output from a REF CURSOR is returned.
- **ResultSetConverter**—An alternative using REF CURSOR.
- **SelectAll**—An outbound Select All operation.
- **SelectAllByTitle**—An outbound SelectAllByTitle operation.
- **SelectCount**—Pure SQL support in 10.1.3.1
- **SelectGroupBy**—Pure SQL support in 10.1.3.1
- **SelectStar**—Pure SQL support in 10.1.3.1
- **sql**—SQL*Server example
- **Update**—A record is passed to the operation and the database row with the same primary key is updated.
- **UpdateAll**—Pure SQL support in 10.1.3.1

See Also: Oracle Adapters for Files, FTP, Databases, and Enterprise Messaging User’s Guide
This samples shows the ability of the JMS Adapter to process incoming messages in a JMS destination (a queue) and write the same message to another JMS destination (a topic).

See Also: Oracle Adapters for Files, FTP, Databases, and Enterprise Messaging User's Guide

These samples show how to use the AQ adapter:

- ADT — A message is received from the AQ adapter, the payload copied to an outbound message, and the AQ adapter invoked with the outbound message. ADT queues are used.
- ADT_with_CLOB_Payload — A message is received from the AQ adapter, the CLOB payload and payload header copied to an outbound message, and the AQ adapter invoked with the outbound message.
- ADT_with_CLOB_Payload_as_Opaque — A process uses a non-XML CLOB field as a payload field.
- ADT_with_XMLType_Payload — A process receives a message from the AQ adapter, copies the payload and PayloadHeader to an outbound message, and invokes the AQ adapter with the outbound message.
- AQ_10_1_3_Supported_AD_Types — Uses SQL Oracle primitive and ANSI types supported by the AQ adapter ADT
- AQMessageRejectionHandler — Rejected messages are handled through the rejectedMessageHandler property.
- AQOutboundCorrelation — Correlations are used to correlate an outbound invoke activity with an inbound receive activity.
- AQSupportedADTTypes — All SQL primitive types supported by the AQ adapter ADT are used.
- File2AQBLOB2File — Reads GIF files from a directory every 10 seconds with the file adapter and enqueues the whole file into a BLOB column field of an ADT queue using the AQ adapter
- MulticonsumerInbound — The AQ adapter listens on the INBOUND_PUBLISHER queue for message recipients named blue. Any message the adapter retrieves starts a BPEL instance. The message is sent to the INBOUND_CONSUMED queue. The queues involved are RAW queues.
- MulticonsumerOutbound — Sets the AQ recipient in a multiconsumer queue. The recipient list is set in the InteractionType parameter or the AQ header recipientList parameter.
- Raw — A message is received from the AQ adapter, the payload copied to an outbound message, and the AQ adapter invoked with the outbound message. RAW queues are used.
- RawQueuePayloadUsingNativeFormat — The AQ Adapter and Native Format Builder wizard are used together. The native format used is comma-separated value (CSV).
- RuleBasedSubscription_Header — A rule-based subscriber is created. The subscriber gets messages and passes them on.
- RuleBasedSubscription_Payload — A rule-based subscriber subscribes to a magazine with a specific title. The message selector rule is used.
- Simple_XMLType_Payload — Simple XMLType payload use

See Also: Oracle Adapters for Files, FTP, Databases, and Enterprise Messaging User's Guide
### Getting Started with Demonstrations, Activity and Conceptual References, and Tutorials

<table>
<thead>
<tr>
<th>Directory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>125.ReportsSchema</td>
<td>This sample shows how to build custom reports using the BPEL Process Manager reports schema.</td>
</tr>
<tr>
<td>126.DataAggregator</td>
<td>This sample shows how to take a single XML document, divide it into several smaller documents, perform tasks on each smaller document, reassemble the smaller documents into a single XML document, and return the single document to the invoker.</td>
</tr>
<tr>
<td>127.OrderBookingTutorial</td>
<td>This sample shows how to design and execute a sophisticated process that uses synchronous and asynchronous services, parallel flows of execution, conditional branching logic, fault handling and exceptions management, transformations, file adapter and database adapter functionality, and human workflow, notification, and sensor functionality.</td>
</tr>
<tr>
<td>128.GoogleFlow</td>
<td>This sample shows a process that uses an external Web service to present information to the client. Processes designed with sensors are also used.</td>
</tr>
</tbody>
</table>
| 129.FTPAdapter | These samples show how to use the file adapter:  
  - FTPDebatching—This sample shows how to use the FTP adapter to process a file containing a batch of business records (invoices and purchase orders) and transform and write the records to separate output files.  
  - SynchronousRead—This sample shows a midprocess synchronous read operation through an invoke activity.  
  See Also: Oracle Adapters for Files, FTP, Databases, and Enterprise Messaging User’s Guide |
| 130.SendEmailWithAttachments | This sample shows how to send an e-mail with attachments through Oracle JDeveloper. |
| 131.UserTasks | This process demonstrate a simple user task. The process has a quote to buy and sell a particular stock and the approver has to select whether to buy or sell the stock. |
| 132.SecureInvokingProcesses | This sample illustrates how to securely invoke a BPEL Process. The following types of clients are covered in this sample:  
  - Invoking from JSP  
  - Invoking from HTTP directly  
  - Invoking over SOAP  
  - Invoking from Java RMI client |
| 140.AdapterFramework | This sample shows how to use dynamic JCA partner links in BPEL. |
| 150.AppsAdapter | These samples show how to use the Oracle Applications adapter:  
  - ChangeOrderAPI—Changes a purchase order in Oracle E-Business Suite  
  - GetPOAckBusinessEvent—Demonstrates outbound business events  
  - OrderImportConcurrentProgram—Imports and creates a purchase order in Oracle E-Business Suite  
  - POAckOutboundXMLGateway—Integrates with Oracle XML gateway to retrieve a purchase order acknowledgement  
  - POInboundXMLGateway—Integrates with Oracle XML gateway to create a purchase order in Oracle E-Business Suite |
| 701.LargeProcesses | This sample shows how support is provided for processes with a large number of work items (10,000 or more). |
This chapter introduces BPEL, how Oracle BPEL Process Manager supports BPEL, and how Oracle JDeveloper enables you to design BPEL processes. An overview of how to use this guide and references to additional tutorials, demonstrations, and other helpful materials installed with Oracle BPEL Process Manager are also provided.

Table 1–5 (Cont.) tutorials Directory Contents

<table>
<thead>
<tr>
<th>Directory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>702. Bindings</td>
<td>This sample shows how to:</td>
</tr>
<tr>
<td></td>
<td>- Integrate Enterprise Java Beans (EJB) in a BPEL process</td>
</tr>
<tr>
<td></td>
<td>- Call the HTTP get method from a BPEL process. This tutorial enables you to call representation state transfer (REST) services from Oracle BPEL Process Manager.</td>
</tr>
<tr>
<td></td>
<td>- Call a Java method from a BPEL process</td>
</tr>
</tbody>
</table>

See Also: The following guides for additional tutorials you can run:

- Oracle BPEL Process Manager Quick Start Guide
- Oracle BPEL Process Manager Order Booking Tutorial
This chapter describes how to start key Oracle BPEL Process Manager components, including Oracle JDeveloper, Oracle BPEL Server, and Oracle BPEL Control. An overview of the main sections of Oracle JDeveloper that you use to design BPEL processes is also provided. Key BPEL design components such as activities and partner links and the services and adapters that Oracle BPEL Process Manager provides to add value and ease of use to standard BPEL functionality are also described.

This chapter contains the following topics:

- **Overview of Oracle BPEL Process Manager Components**
- **Starting Oracle BPEL Process Manager Components**
- **Overview of the BPEL Designer Environment**
- **Overview of Activities**
- **Overview of Partner Links**
- **Overview of Oracle BPEL Server**
- **Overview of Oracle BPEL Control**
- **Overview of Oracle BPEL Process Manager Services**
- **Overview of Oracle BPEL Process Manager Technology Adapters**
- **Summary**

**Overview of Oracle BPEL Process Manager Components**

The Oracle BPEL Process Manager consists of the three components shown in Figure 2-1.

![Figure 2-1 Oracle BPEL Process Manager Components](image)

Each component enables you to perform a specific set of tasks:
Starting Oracle BPEL Process Manager Components

- The design environment (Oracle JDeveloper) enables you to design and deploy BPEL processes. You design BPEL processes by dragging and dropping elements (known as activities) into the process and editing their property pages. You integrate BPEL processes with external services that you also design and edit (known as partner links). You also integrate technology adapters and services such as workflows, worklists, transformations, notifications, sensors, and business rules with the process.
- When design is complete, you deploy the process from the design environment to Oracle BPEL Server.
- If deployment is successful, you can run and manage the BPEL process from Oracle BPEL Control.

This chapter provides an overview of getting started with these components.

### Starting Oracle BPEL Process Manager Components

Follow the instructions in Table 2–1 to start and stop Oracle BPEL Process Manager components.

<table>
<thead>
<tr>
<th>To Access The...</th>
<th>On Windows...</th>
<th>On UNIX...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oracle BPEL Server</td>
<td>Select Start &gt; All Programs &gt; Oracle - Oracle_Home &gt; Start SOA suite</td>
<td>To start Oracle BPEL Server: From $ORACLE_HOME/bpel/bin: startorabpel.sh To stop Oracle BPEL Server: From $ORACLE_HOME/bpel/bin: shutdownorabpel.sh</td>
</tr>
<tr>
<td>Oracle JDeveloper</td>
<td>Click JDev_Oracle_Home\JDev\bin\jdev.exe or create a shortcut</td>
<td>$ORACLE_HOME/jdev/bin/jdev</td>
</tr>
<tr>
<td>Oracle BPEL Control</td>
<td>You must first start Oracle BPEL Server. To start Oracle BPEL Control: 1. Select Start &gt; All Programs &gt; Oracle - Oracle_Home &gt; Oracle BPEL Process Manager &gt; BPEL Control You can also start Oracle BPEL Control using the URL for your installation, which can be found in SOA_Oracle_Home\install\bpelsetupinfo.txt.</td>
<td>First start Oracle BPEL Server. To start Oracle BPEL Control: ■ Log on to the URL for your installation, which can be found in bpelsetupinfo.txt.</td>
</tr>
</tbody>
</table>
Overview of the BPEL Designer Environment

This section provides an overview of the Oracle JDeveloper environment.

Overview of BPEL Project Creation and Oracle JDeveloper

This section provides an overview of Oracle JDeveloper. In this overview, you first create an application and a project. An application is a container in which to place projects. A project contains the BPEL process.

1. Create an application by selecting New > Application from the File main menu and providing the required details in the Create Application window (including not selecting any application template).
2. Ensure that the directory path of an application does not include any blank spaces. For example, the following is not permitted:
   ```
   C:\Program Files\projects\myapplication\Loanflow
   ```
3. Click Cancel on the Create Project window.
4. Right-click the newly created application and select New Project.
5. Double-click BPEL Process Project and provide the required details (including BPEL process name) in the BPEL Project Creation Wizard windows. A single
A project can contain only one BPEL process. Always use completely unique names when creating BPEL projects. Do not create:

- A project name that begins with a number
- A project name that includes a dash (for example, Loan-Flow)
- Two projects with the same name, but with different capitalization

**Notes:**

- You can also import existing projects into Oracle JDeveloper by selecting Import > BPEL Process from the File main menu. However, do not import or add XSD files in a ZIP file into a BPEL project. Always extract the XSD files from a ZIP file before importing them.
- You can also create and select templates of existing projects. Templates enable you to make copies of BPEL processes. For example, assume you first design a large and complicated BPEL process. You then need to design a second BPEL process that is very similar, but not quite the same, as the previous BPEL process. Instead of designing the second process completely from scratch, you can create a template of the first process, then select it when creating a project for the second BPEL process. This creates the same BPEL process as the first one. You can then modify the second process as necessary.

Right click a project, select Mark as Template, and specify a name. When you create a new BPEL project, you can select the process name from the Templates list of the BPEL Project Creation Wizard.

After you create the application and project, Oracle JDeveloper displays the sections shown in Figure 2-2. You can also access this view by selecting View > Application Navigator and double-clicking the .bpel file of the project. In this example, the project is an asynchronous type and is named OrderBooking.
Overview of the BPEL Designer Environment

Each section of this view enables you to perform specific design and deployment tasks. Table 2–2 identifies the sections listed in Figure 2–2 and provides references to sections that describe their capabilities.

Table 2–2 Oracle JDeveloper Sections

<table>
<thead>
<tr>
<th>Section</th>
<th>Location in Figure 2–2</th>
<th>See Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application Navigator</td>
<td>Upper left</td>
<td>“Application Navigator” on page 2-6</td>
</tr>
<tr>
<td>Diagram window, Source window, and History window</td>
<td>Middle</td>
<td>“Diagram Window” on page 2-7, “Source Window” on page 2-8, and “History Window” on page 2-10</td>
</tr>
<tr>
<td>Process Activities selection of the Component Palette</td>
<td>Upper right</td>
<td>“Component Palette” on page 2-10</td>
</tr>
<tr>
<td>Property Inspector section</td>
<td>Lower right</td>
<td>“Property Inspector” on page 2-12</td>
</tr>
<tr>
<td>Structure Window</td>
<td>Lower left</td>
<td>“Structure Window” on page 2-12</td>
</tr>
<tr>
<td>Log Window</td>
<td>Bottom</td>
<td>“Log Window” on page 2-13</td>
</tr>
</tbody>
</table>

See Also: Oracle BPEL Process Manager Quick Start Guide and Oracle BPEL Process Manager Order Booking Tutorial for tutorials in which you create applications and projects.
Application Navigator

The Application Navigator shown in the upper left part of Figure 2–2 displays the project files. Double-click a node (for example, the Integration Content node) to display its contents. Right-click a node to display a context-sensitive menu of commands. The menu commands that are available depend on the node selected. For example, if you right-click the FulfillOrders project in Figure 2–3, you can compile and deploy this BPEL process to Oracle BPEL Server.

Figure 2–3 shows the files that appear under the Integration Content folder when you first create a project in Oracle JDeveloper (in this example, named FulfillOrders inside an application named myBPELapplication).

Figure 2–3 Application Navigator

Table 2–3 describes these initial project files.

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bpel.xml</td>
<td>The deployment descriptor file that defines the locations of the WSDL files for services to be called by this BPEL process flow. This file references the public interface for the service.</td>
</tr>
<tr>
<td></td>
<td>See Also: Appendix C, “Deployment Descriptor Properties”</td>
</tr>
<tr>
<td>FulfillOrder.bpel</td>
<td>The source file, which, depending upon the project type you selected, initially contains a minimal set of activities (if you selected to create an asynchronous project, then receive and invoke activities appear). You add syntax to this file when you drag and drop activities, create variables, create partner links, and so on.</td>
</tr>
<tr>
<td>FulfillOrder.wsdl</td>
<td>The WSDL client interface, which defines the input and output messages for this BPEL process flow, the supported client interface and operations, and other features. This functionality enables the BPEL process flow to be called as a service.</td>
</tr>
</tbody>
</table>

As you design the project, additional files, folders, and elements can appear in the Application Navigator. For example, Figure 2–4 shows the files that appear for a project in which you imported schemas (OrderBookingPO.xsd and Orders.xsd), configured the database adapter (the WriteDBRecord.wsdl file), and created a transform activity (Transformation_1.xsl under the Integration Content folder). The Application Sources node contains Java source files. The Java classes are used inside callouts from the BPEL process. Additional folders can appear, such as BPEL-INF (a special directory for Java JAR files).
The Diagram window shown in the middle of Figure 2–2 provides a visual view of the BPEL process that you design. This view displays when you perform one of the following actions:

- Double-click the .bpel file name in the Application Navigator
- Click the Diagram tab at the bottom of the window with the .bpel file selected

Figure 2–5 shows the activities automatically created with an asynchronous project. In the tutorials described in Oracle BPEL Process Manager Quick Start Guide and Oracle BPEL Process Manager Order Booking Tutorial, you add to the BPEL process by dragging and dropping activities, creating variables, creating partner links, and so on.
As you design the project by dragging and dropping activities, creating partner links, and so on, the Diagram window changes. Figure 2–6 shows the Diagram window later in the design phase after adding a partner link (in this example, named WriteDBRecord) and the additional activities (invoke, receive, assign, transform, and others).

Figure 2–6  Diagram (After Design Phase)

Source Window
Click Source at the bottom to view the syntax inside the BPEL process project files. As you drag and drop activities and partner links, and perform other tasks, the syntax in these source files is immediately updated to reflect these changes. For example, Figure 2–7 shows the property sheet as it is being edited.
Figure 2–7  CreditRatingService Partner Link Icon and Property Sheet

Click Source at the bottom of the window. Figure 2–8 shows part of the Source of a .bpel file. Details about the CreditRatingService partner link you created appear in the file.

Figure 2–8  Source View of a .bpel File

```xml
<partnerlinks>
  <partnerlink name="client" partnertype="client:CreditRatingService">
    <bpel:partnerlink/>
    <partnerRole>"my:CreditRatingRequester"</partnerRole>
  </partnerlink>
  <partnerlink name="CreditRatingService" partnertype="all:CreditRatingService">
    <partnerRole>"CreditRatingServiceSupplier"</partnerRole>
    <partnertype>"all:CreditRatingService"</partnertype>
  </partnerlink>
</partnerlinks>
```
History Window

Click History at the bottom to perform such tasks as viewing the revision history of a file and viewing read-only and editable versions of a file side-by-side. Figure 2–9 shows the History view for a BPEL file.

![History View](image)

Note: If you want to learn more about the History view, place the cursor in this section and press F1 to display online Help.

Component Palette

Activities are the building blocks of the BPEL process. The Process Activities selection of the Component Palette shown in the upper right part of Figure 2–2 displays a set of activities that you drag and drop into the Diagram window of the BPEL process. The Component Palette displays only those pages relevant to the state of the Diagram window. Process Activities or Services are nearly always visible. However, if you are designing a transformation in a transform activity, the Component Palette only...
displays selections relevant to that activity, such as String Functions, Mathematical Functions, and Node-set Functions.

Figure 2–10 shows the Process Activities selection of the Component Palette. This list enables you to select activities to drag and drop into your BPEL process.

Figure 2–10 Component Palette - Process Activities

Figure 2–11 shows the Services selection of the Component Palette. This list enables you to drag and drop adapters, partner links, or decision services into your BPEL process.

Figure 2–11 Component Palette - Services

Figure 2–12 shows the String Functions category of the Component Palette that displays when you work in the transformation window of a transform activity.
Overview of the BPEL Designer Environment

**Figure 2–12 Component Palette - Functions**

![Component Palette - Functions](image)

**Note:** If you want to learn more about the Component Palette, place the cursor in this section and press F1 to display online Help.

**Property Inspector**

The Property Inspector shown in the lower right part of Figure 2–2 enables you to view details about an activity. Single-click an activity in the Diagram window. For example, single-clicking the receiveInput receive activity shown in Figure 2–5 on page 2-7 displays the information shown in Figure 2–13.

**Figure 2–13 Property Inspector**

![Property Inspector](image)

**Structure Window**

The Structure Window shown in the lower left part of Figure 2–2 offers a structural view of the data in the project currently selected in the Diagram window. You can perform a variety of tasks from this section, including:

- Importing project schemas
- Defining message types
- Managing (creating, editing, and deleting) elements such as variables, aliases, correlation sets, partner links, and sensors
- Editing activities in the BPEL process flow sequence that displays in the Diagram window
Figure 2–14 shows the **Structure Window**. In this example, the window has been expanded to display the imported project schemas and the sequence of activities in the **Diagram window** for an OrderBooking project.

**Figure 2–14 Structure Window (Expanded)**

Log Window

You validate, compile, and deploy a process by right-clicking the project name in the **Application Navigator**, selecting **Deploy**, and selecting a deployment method. The **Log Window** shown at the bottom of Figure 2–2 then displays messages about the status of the deployment.

To ensure that a process validates correctly, you must ensure that the following information is correct:

- The process must have an input variable.
- A partner link must be selected.
- A partner role must be selected.
- The operation must not be empty.
- The input variable type must match the partner link operation type.

**Figure 2–15** shows a successful deployment message for a BPEL process. You can then run, monitor, and administer the process from Oracle BPEL Control.

---

**Notes:**

- If you want to learn more about the **Structure Window**, place the cursor in this section and press **F1** to display online Help.
- Do not import two schema files with the same name into a project. Ensure that the files have unique names.
If deployment is unsuccessful, messages appear that describe the type and location of the error, as shown in Figure 2–16. Double-click the error to navigate directly to the offending line in the source file referenced.

Note: If you want to learn more about the Log Window, place the cursor in this section and press F1 to display online Help.

See Also:
- "Overview of Oracle BPEL Control" on page 2-17
- Chapter 19, ‘BPEL Process Deployment and Domain Management’ for specific details about deploying and running BPEL processes

Editing Project Files in Oracle JDeveloper
Note the following issues when editing the bpel.xml, WSDL, and BPEL files:

- The bpel.xml file content is only read into memory when the file is opened. Therefore, if you change the content of bpel.xml after the file is opened, the changes are not made in memory. After changing the content of the BPEL file, close and reopen the file for the changes to take effect.

- Do not edit the bpel.xml file through a combination of Oracle JDeveloper and a text editor such as Notepad or Wordpad. Use only a single editing environment such as Oracle JDeveloper.

- Do not edit the bpel.xml file, BPEL files, and WSDL files while changing the design of the process. If you want to edit a file:
  1. Ensure that the BPEL files are not being edited in Oracle JDeveloper. If they are being edited (that is, a tab for that file is visible), close it and save changes as needed.
  2. Edit the required file and save the changes.
Overview of Activities

The term *activities* has been mentioned frequently in both Chapter 1, "Introduction to Oracle BPEL Process Manager" and in this chapter. Activities are the building blocks of a BPEL process. Oracle JDeveloper includes a set of activities that you drag and drop into a BPEL process. You then double-click an activity to define its attributes (property values). Figure 2–6 on page 2-8 provides an example of this design process. Activities enable you to perform specific tasks within a process. For example:

- An assign activity enables you to manipulate data, such as copying the contents of one variable to another.

- An invoke activity enables you to invoke a service (identified by its partner link) and specify an operation for this service to perform.

- A receive activity waits for an asynchronous callback response message from a service.

Figure 2–17 shows an example of a property window (for this example, an invoke activity). In this example, you invoke a partner link named *Invoke_FileWrite* and define its attributes.

*Figure 2–17 Invoke Activity Example*
Overview of Partner Links

The term partner link has also been mentioned frequently in both Chapter 1, “Introduction to Oracle BPEL Process Manager” and in this chapter. A partner link enables you to define the external services with which the BPEL process is to interact.

Figure 2–18 shows the partner link icon (in this example, named CreditRating).

Figure 2–18  PartnerLink Icon

A partner link type characterizes the conversational relationship between two services by defining the roles played by each service in the conversation and specifying the port type provided by each service to receive messages within the context of the conversation. Figure 2–16 on page 2-8 shows an example of a partner link named WriteDBRecord being invoked by a BPEL process.

Figure 2–19 shows an example of the attributes of a partner link for a service named CreditRating.

See Also:

- Appendix B, ‘BPEL Process Activities and Services’ for descriptions of available activities
- Part II, ‘Reviewing Key BPEL Development Concepts and Code Samples’ for activity concepts and code examples
- $ORA_HOME/bpel/samples/references directory for additional activity code examples
- Oracle BPEL Process Manager Quick Start Guide and Oracle BPEL Process Manager Order Booking Tutorial for tutorials in which you drag and drop activities in BPEL processes and define their attributes

Overview of Partner Links

The term partner link has also been mentioned frequently in both Chapter 1, “Introduction to Oracle BPEL Process Manager” and in this chapter. A partner link enables you to define the external services with which the BPEL process is to interact.

Figure 2–18 shows the partner link icon (in this example, named CreditRating).

Figure 2–18  PartnerLink Icon

A partner link type characterizes the conversational relationship between two services by defining the roles played by each service in the conversation and specifying the port type provided by each service to receive messages within the context of the conversation. Figure 2–16 on page 2-8 shows an example of a partner link named WriteDBRecord being invoked by a BPEL process.

Figure 2–19 shows an example of the attributes of a partner link for a service named CreditRating.
Overview of Oracle BPEL Server

After you complete the design of the BPEL process, you compile and deploy the process to Oracle BPEL Server. If compilation and deployment are successful, you can run and manage the BPEL process from Oracle BPEL Control.

Deployment sends the Oracle BPEL Process Manager archive (a set of files in a JAR file with a directory structure similar to the project directory structure) to Oracle BPEL Server. The deployment operation automatically validates and compiles the project directory into the BPEL archive.

See Also: Chapter 19, "BPEL Process Deployment and Domain Management"

Overview of Oracle BPEL Control

Oracle BPEL Control enables you to run, monitor, and administer BPEL processes designed and deployed with Oracle JDeveloper. You can also manage BPEL domains.
Overview of Oracle BPEL Process Manager Services

from Oracle BPEL Control. Access Oracle BPEL Control on Windows by selecting Start > All Programs > Oracle - Oracle_Home > Oracle BPEL Process Manager > BPEL Control.

Figure 2–20 shows the main page of Oracle BPEL Control. In this example, a number of deployed BPEL processes and external services appear in the Dashboard tab.

Table 2–5 identifies and describes the services and provides references to sections of this guide that describe their capabilities.

See Also:
- “Starting Oracle BPEL Process Manager Components” on page 2-2 for instructions on accessing Oracle BPEL Control on UNIX
- Chapter 19, "BPEL Process Deployment and Domain Management" for specific details about running a deployed process from Oracle BPEL Control
- Oracle BPEL Process Manager Quick Start Guide and Oracle BPEL Process Manager Order Booking Tutorial for tutorials in which you run deployed BPEL processes

Overview of Oracle BPEL Process Manager Services

Oracle BPEL Process Manager and Oracle JDeveloper provide support for services that add value and ease of use to BPEL functionality.

Table 2–5 identifies and describes the services and provides references to sections of this guide that describe their capabilities.
## Table 2–5  Oracle BPEL Process Manager Services

<table>
<thead>
<tr>
<th>Types</th>
<th>Description</th>
<th>See Section</th>
</tr>
</thead>
</table>
| Transformations        | A transform activity is provided that enables you to create transformations that map source data to target data. For example, you can map incoming purchase order source data into outgoing purchase order acknowledgment target data. | Chapter 13, “XSLT Mapper and Transformations”  
“Transform Activity” on page B-29 |
| Notification channels  | Notification channels enable you to send notifications about an event to a user, group, or destination address. You can send a notification by e-mail, voice mail, fax, pager, or short message service (SMS).          | Chapter 14,  
“Oracle BPEL Process Manager Notification Service”  
Appendix B,  
“BPEL Process Activities and Services” |
| Workflows              | Workflow enables you to integrate systems and services with human workflow into a single process flow. A Human Task editor is provided that enables you to specify human task settings, such as task outcome, payload structure, task participants, assignment and routing policy, expiration and escalation policy, notification settings, and so on. The criteria that you define with the Human Task editor enables you to use the Oracle BPEL Worklist Application when you run the BPEL process. | Chapter 15,  
“Oracle BPEL Process Manager Workflow Services”  
“Human Task Activity” on page B-12 |
| Oracle BPEL Worklist Application | Oracle BPEL Worklist Application takes actions on tasks such as approving an employee vacation request, evaluating a job applicant, or escalating a purchasing decision. Based on the user profile, you access a URL that enables you to see all the tasks relevant to you and specify search criteria for displaying tasks. | Chapter 16,  
“Worklist Application” |
| Sensors                | You create sensors that you assign to activities, variables, and faults that you want to monitor during BPEL process run time.                                                                                     | Chapter 17,  
“Sensors” |
| Business rules         | You integrate BPEL processes with the rules defined in a business rule engine.                                                                                                                              | Chapter 18,  
“BPEL Process Integration with Business Rules” |

See Also: The following documentation for tutorials that describe how to design BPEL processes that use the services described in Table 2–5

- Oracle BPEL Process Manager Order Booking Tutorial
- “Getting Started with Demonstrations, Activity and Conceptual References, and Tutorials” on page 1-6
Overview of Oracle BPEL Process Manager Technology Adapters

The Partner Link Window shown in Figure 2–19 on page 2-17 also enables you to take advantage of another key feature that Oracle BPEL Process Manager and Oracle JDeveloper provide. Click the Define Adapter Service icon shown in Figure 2–21 to access the Adapter Configuration wizard.

Adapters enable you to integrate the BPEL processes with access to file systems, FTP servers, database tables, database queues, Java Message Services (JMS), MQ, and Oracle E-Business Suite. This wizard enables you to configure the types of adapters shown in Figure 2–22 for use with the BPEL process:

When you select an adapter type, the Service Name window shown in Figure 2–23 prompts you to enter a name. For this example, File Adapter was selected in Figure 2–22. When the wizard completes, a WSDL file by this service name appears in the Application Navigator for the BPEL process (for this example, named ReadFile.wsdl). This file includes the adapter configuration settings you specify with this wizard. Other configuration files (such as header files and files specific to the adapter) are also created and display in the Application Navigator.
The Adapter Configuration wizard windows that appear after the Service Name window are based on the adapter type you selected.

**See Also:**

- Oracle Adapters for Files, FTP, Databases, and Enterprise Messaging User’s Guide for specific details about configuring the file, FTP, database, AQ, MQ, and JMS adapters in a BPEL process with the Adapter Configuration wizard
- Oracle Application Server Adapter for Oracle Applications User’s Guide for information on using the Oracle Applications adapter for Oracle E-Business Suite
- “PartnerLink” on page B-36
- Oracle BPEL Process Manager Order Booking Tutorial for tutorials that describe how to design BPEL processes that use the database adapter and the file read and write functionality of the file adapter

**Summary**

This chapter describes how to start key Oracle BPEL Process Manager components, including Oracle JDeveloper, Oracle BPEL Server, and Oracle BPEL Control. An overview of the main sections of Oracle JDeveloper that you use to design BPEL processes is also provided. Key BPEL design components such as activities and partner links and the services and adapters that Oracle BPEL Process Manager provides to add value and ease of use to standard BPEL functionality are also described.
Part II

Reviewing Key BPEL Development Concepts and Code Samples

This part introduces key BPEL development concepts and code samples. This part contains the following chapters:

- Chapter 3, "Manipulating XML Data in BPEL"
- Chapter 4, "Invoking a Synchronous Web Service"
- Chapter 5, "Invoking an Asynchronous Web Service"
- Chapter 6, "Parallel Flow"
- Chapter 7, "Conditional Branching"
- Chapter 8, "Fault Handling"
- Chapter 9, "Incorporating Java and J2EE Code in BPEL Processes"
- Chapter 10, "Events and Timeouts"
- Chapter 11, "Invoking a BPEL Process"
- Chapter 12, "Interaction Patterns"
This chapter describes how to manipulate XML data in BPEL, including the use of XPath expressions. This chapter contains the following topics:

- Use Cases for Manipulating XML Data in BPEL
- Overview of Manipulating XML Data in BPEL Concepts
- Initializing a Variable with Expression Constants or Literal XML
- Copying Between Variables
- Accessing Fields Within Element-Based and Message Type-Based Variables
- Assigning Numeric Values
- Mathematical Calculations with XPath Standards
- Assigning String Literals
- Concatenating Strings
- Assigning Boolean Values
- Assigning Date or Time
- Manipulating Attributes
- Manipulating XML Data with bpelx Extensions
- Validating XML Data with bpelx:validate
- Manipulating XML Data Sequences That Use Arrays
- Converting from a String to an XML Element
- Differences Between Document-Style and RPC-Style WSDL Files
- Adding a Custom WSIF Provider
- Input and Output Message Header Handling
- Manipulating SOAP Headers in BPEL
- Using Binary Attachments in SOAP Messages
- Summary

Use Cases for Manipulating XML Data in BPEL

This chapter covers a variety of use cases for manipulating XML data. Topics include how to work with variables, sequences, and arrays, and how to perform tasks such as...
Overview of Manipulating XML Data in BPEL Concepts

This section covers the following topics:

- How XML Data Works in BPEL
- About Data Manipulation and XPath Standards

How XML Data Works in BPEL

In a BPEL process, every piece of data is in XML forms. This includes the messages passed to and from the BPEL process, the messages exchanged with external services, and local variables used by the process. You define the types for these messages and variables with the XML schema, usually in the WSDL file for the flow, the WSDL files for the services it invokes, or the XSD file referenced by those WSDL files. Therefore, all variables in BPEL are XML data, and any BPEL process uses much of its code to manipulate these XML variables. This typically includes performing data transformation between representations required for different services, and local manipulation of data (for example, to combine the results from several service invocations).

About Data Manipulation and XPath Standards

The starting point for data manipulation in BPEL is the assign activity, which builds on the XPath standard. XPath queries, expressions, and functions play a large part in this type of manipulation. In addition, more advanced methods are available that involve using XQuery, XSLT, or Java, usually to do more complex data transformation or manipulation.

This section provides a general overview of how to manipulate XML data in BPEL. It summarizes the key building blocks used in various combinations and provides examples. The remaining sections in this chapter discuss and illustrate how to apply these building blocks to perform specific tasks.

You use the assign activity to copy data from one XML variable to another, or to calculate the value of an expression and store it in a variable. A copy element within the activity specifies the source and target of the assignment (what to copy from and to), which must be of compatible types. The formal syntax as shown in the Business Process Execution Language for Web Services Specification is as follows:

```xml
<assign standard-attributes>
  standard-elements
  <copy>+
    from-spec
to-spec
  </copy>
</assign>
```

Mathematical calculations. The explanations are largely by example, and provide an introduction to the supported specifications.

Most of the examples in this chapter assume that the WSDL file defining the associated message types is document-literal style rather than the RPC style. There is a difference in how XPath query strings are formed for RPC-style WSDL definitions. If you are working with a type defined in an RPC WSDL file, see “Differences Between Document-Style and RPC-Style WSDL Files” on page 3-22.

See Also: The sample files located at

- SOA_Oracle_Home\bpel\samples\tutorials\103.XMLDocuments
This syntax is described in detail in that specification. The from-spec and to-spec typically specify a variable or variable part, as in:

```xml
<assign>
  <copy>
    <from variable="c1" part="address"/>
    <to variable="c3"/>
  </copy>
</assign>
```

When you use Oracle JDeveloper, you supply assign activity details in a Copy Operation window that includes a From section and a To section. This reflects the preceding BPEL source code syntax.

Rather than repeating all syntax details, this chapter shows and describes excerpts taken primarily from sample projects provided in the SOA_Oracle_Home\bpel\samples\references directory.

XPath standards play a key role in the assign activity. Brief examples are shown here as an introduction; examples with more context and explanation are provided in the sections that follow.

- **XPath queries**: An XPath query selects a field within a source or target variable part. The from or to clause can include a query attribute whose value is an XPath query string. For example:
  ```xml
  <from variable="input" part="payload" query="/p:CreditFlowRequest/p:ssn"/>
  ```

  For XPath version 1.0, the value of the query attribute must be a location path that selects exactly one node. You can find further details about the query attribute and XPath standards syntax in the Business Process Execution Language for Web Services Specification (section 14.3) and the XML Path Language (XPath) Specification, respectively.

- **XPath expressions**: You use an XPath expression (specified in an expression attribute in the from clause) to indicate a value to be stored in a variable. For example:
  ```xml
  <from expression="100"/>
  ```

  The expression can be any general expression—that is, an XPath expression that evaluates to any XPath value type. For more information about XPath expressions, see section 9.1.4 of the XML Path Language (XPath) Specification.

Within XPath expressions, you can call the following types of functions:

- **Core XPath functions**: XPath supports a large number of built-in functions, including functions for string manipulation (such as concat), numeric functions (like sum), and others.

  ```xml
  <from expression="concat('string one', 'string two')"/>
  ```

  For a complete list of the functions built into XPath standards, see section 4 of the XML Path Language (XPath) Specification.

- **BPEL XPath extension functions**: BPEL adds several extension functions to the core XPath core functions, enabling XPath expressions to access information from a process. The extensions are defined in the standard BPEL namespace.
Initializing a Variable with Expression Constants or Literal XML

It is often useful to assign literal XML to a variable in BPEL, for example, to initialize a variable before copying dynamic data into a specific field within the XML data content for the variable. This is also useful for testing purposes when you want to hard code XML data values into the process.

Oracle BPEL XPath extension functions: Oracle provides some additional XPath functions that use the capabilities built into BPEL and XPath standards for adding new functions. These functions are defined in the namespace http://schemas.oracle.com/xpath/extension and indicated by the prefix ora:.

See Also: Appendix D, "XPath Extension Functions"

Custom functions: You can also create custom XPath functions. If you do, you must register them in the BPEL process deployment descriptor or in the following XML files:
- SOA_Oracle_Home\bpel\system\config\xpath-functions.xml (system level)
- SOA_Oracle_Home\bpel\domains\default\config\xpath-functions.xml (domain level)

Then, package the source implementing them into a BPEL suitcase or Oracle BPEL Process Manager startup environment. For more information about writing custom XPath functions, refer to:

http://www.oracle.com/technology/bpel

Sophisticated data manipulation can be difficult to perform with the BPEL assign activity and the core XPath functions. However, you can perform complex data manipulation and transformation by using XSLT, Java, or a bpelx operation under an assign activity (See "Manipulating XML Data with bpelx Extensions" on page 3-10), or as a Web service. For more information on calling Java code from within BPEL, see the tutorial under the BPEL Tutorials link at http://www.oracle.com/technology/bpel. For XSLT, Oracle BPEL Process Manager includes XPath functions that execute these transformations.

See Also: The following XPath and XQuery transformation code examples:
- SOA_Oracle_Home\bpel\samples\tutorial\114.XSLTTransformations
- Chapter 13, "XSLT Mapper and Transformations"

The following sections show related definitions in the BPEL and WSDL files that help explain the examples.
This example assigns a literal `<result>` element to the payload part of the `output` variable:

```xml
<assign>
  <!-- copy from literal xml to the variable -->
  <copy>
    <from>
      <result xmlns="http://samples.otn.com">
        <name/>
        <symbol/>
        <price>12.3</price>
        <quantity>0</quantity>
        <approved/>
        <message/>
      </result>
    </from>
    <to variable="output" part="payload"/>
  </copy>
</assign>
```

See Also: The following samples:
- SOA_Oracle_Home\bpel\samples\references\Assign

### Copying Between Variables

When you copy between variables, you copy directly from one variable (or part) to another variable of a compatible type, without needing to specify a particular field within either variable. In other words, there is no need to specify an XPath query.

The following example performs two assignments, first copying between two variables of the same type and then copying a variable part to another variable with the same type as that part.

```xml
<assign>
  <copy>
    <from variable="c1"/>
    <to variable="c2"/>
  </copy>
  <copy>
    <from variable="c1" part="address"/>
    <to variable="c3"/>
  </copy>
</assign>
```

The BPEL file defines the variables as follows:

```xml
<variable name="c1" messageType="x:person"/>
<variable name="c2" messageType="x:person"/>
<variable name="c3" element="x:address"/>
```

The WSDL file defines the person message type as follows:

```xml
<message name="person" xmlns:x="http://tempuri.org/bpes/example">
  <part name="full-name" type="xsd:string"/>
  <part name="address" element="x:address"/>
</message>
```

See Also: Section 9.3.2 of the Business Process Execution Language for Web Services Specification for this code example
Accessing Fields Within Element-Based and Message Type-Based Variables

Given the types of definitions present in most WSDL and XSD files, you must go down to the level of copying from or to a field within part of a variable based on the element and message type, which in turn uses XML schema complex types. To do this, you specify an XPath query in the from or to clause of the assign activity.

This example copies the ssn field from the CreditFlow process’s input message into the ssn field of the credit rating service’s input message.

```xml
<assign>
  <copy>
    <from variable="input" part="payload" query="/tns:CreditFlowRequest/tns:ssn"/>
    <to variable="crInput" part="payload" query="/tns:ssn"/>
  </copy>
</assign>
```

The BPEL file defines the variables involved in this assignment as follows:

```xml
<variable name="input" messageType="tns:CreditFlowRequestMessage"/>
<variable name="crInput" messageType="services:CreditRatingServiceRequestMessage"/>
```

The crInput variable is used as an input message to a credit rating service. Its message type, CreditFlowRequestMessage, is defined in CreditFlowService.wsdl as follows:

```xml
<message name="CreditFlowRequestMessage">
  <part name="payload" element="tns:CreditFlowRequest"/>
</message>
```

CreditFlowRequest is defined with a field named ssn. The message type CreditRatingServiceRequestMessage is defined in CreditRatingService.wsdl as follows:

```xml
<message name="CreditRatingServiceRequestMessage">
  <part name="payload" element="tns:ssn"/>
</message>
```

See Also: The following sample:

- SOA_Oracle_Home\bpel\samples\utils\CreditRatingService

Assigning Numeric Values

You can assign numeric values in XPath expressions. The following example shows how to assign an XPath expression with the integer value 100.

```xml
<assign>
  <!-- copy from integer expression to the variable -->
  <copy>
    <from expression="100"/>
    <to variable="output" part="payload" query="/p:result/p:quantity"/>
  </copy>
</assign>
```
Mathematical Calculations with XPath Standards

You can use simple mathematical expressions like the one in the following example, which increments a numeric value.

In this example, the BPEL XPath function `getVariableData` retrieves the value being incremented. The arguments to `getVariableData` are equivalent to the variable, part, and query attributes of the `from` clause (including the last two arguments, which are optional).

```xml
<assign>
  <copy>
    <from expression="bpws:getVariableData('input', 'payload', '/p:value') + 1"/>
    <to variable="output" part="payload" query="/p:result"/>
  </copy>
</assign>
```

You can also use `$variable` syntax:

```xml
<assign>
  <copy>
    <from expression="$input.payload + 1"/>
    <to variable="output" part="payload" query="/p:result"/>
  </copy>
</assign>
```

See Also: The following sample:

- `SOA_Oracle_Home\bpel\samples\references\Assign`

Assigning String Literals

This example copies an expression evaluating from the string literal 'GE' to the symbol field within the indicated variable part. (Note the use of the double and single quotes.)

```xml
<assign>
  <!-- copy from string expression to the variable -->
  <copy>
    <from expression="'GE'"/>
    <to variable="output" part="payload" query="/p:result/p:symbol"/>
  </copy>
</assign>
```

See Also: The following sample:

- `SOA_Oracle_Home\bpel\samples\references\Assign`

Concatenating Strings

Rather than copy the value of one string variable (or variable part or field) to another, you first can perform string manipulation, such as concatenating several strings together. An example is shown in the following syntax. The concatenation is accomplished with the core XPath function named `concat`; in addition, the variable
Assigning Boolean Values

value involved in the concatenation is retrieved with the BPEL XPath function `getVariableData`.

In this example, `getVariableData` fetches the value of the name field from the input variable's payload part. The string literal 'Hello ' is then concatenated to the beginning of this value.

```xml
<assign>
  <!-- copy from XPath expression to the variable -->
  <copy>
    <from expression="concat('Hello ',
                             bpws:getVariableData('input', 'payload', '/p:name'))"/>
    <to variable="output" part="payload" query="/p:result/p:message"/>
  </copy>
</assign>
```

Other string manipulation functions available in XPath are listed in section 4.2 of the XML Path Language (XPath) Specification.

Assigning Boolean Values

In this example of assigning Boolean values, the XPath expression in the `from` clause is a call to XPath's Boolean function `true`, and the specified approved field is set to `true`. The function `false` is also available.

```xml
<assign>
  <!-- copy from boolean expression function to the variable -->
  <copy>
    <from expression="true()"/>
    <to variable="output" part="payload" query="/result/approved"/>
  </copy>
</assign>
```

The XPath specification recommends that you use the `true()` and `false()` functions as a method for returning Boolean constant values.

If you instead use `boolean(true)` or `boolean(false)`, the `true` or `false` inside the Boolean function is interpreted as a relative element step, and not as any true or false constant. This means it attempts to select a child node named `true` under the current XPath context node. In most cases, the `true` node does not exist. Therefore, an empty result node set is returned and the `boolean()` function in XPath 1.0 converts an empty node set into a false result. This result can be potentially confusing.

```xml
<assign>
  <!-- copy from boolean expression function to the variable -->
  <copy>
    <from expression="true()"/>
    <to variable="output" part="payload" query="/result/approved"/>
  </copy>
</assign>
```

Assigning Date or Time

You can assign the current value of a date or time field by using the Oracle BPEL XPath function `getCurrentDate`, `getCurrentTime`, or `getCurrentDateTime`, respectively. In addition, if you have a date-time value in the standard XSD format, you can convert it to characters more suitable for output by calling the Oracle BPEL XPath function `formatDate`.

See Also: The following sample:

- SOA_Oracle_Home\bpel\samples\references\Assign
Manipulating Attributes

For related information, see section 9.1.2 of the Business Process Execution Language for Web Services Specification.

<!-- execute the XPath extension function getCurrentDate() -->
<assign>
  <copy>
    <from expression="ora:getCurrentDate()"/>
    <to variable="output" part="payload" query="/invoice/invoiceDate"/>
  </copy>
</assign>

In the next example, the formatDate function converts the date-time value provided in XSD format to the string 'Jun 10, 2005' (and assigns it to the string field formattedDate).

<!-- execute the XPath extension function formatDate() -->
<assign>
  <copy>
    <from expression="ora:formatDate('2005-06-10T15:56:00', 'MMM dd, yyyy')"/>
    <to variable="output" part="payload" query="/invoice/formattedDate"/>
  </copy>
</assign>

See Also: The following sample:

■ SOA_Oracle_Home\bpel\samples\references\XPathFunction

Manipulating Attributes

You may want to copy to or from something defined as an XML attribute. An at sign (@) in XPath query syntax refers to an attribute instead of a child element.

The following code example fetches and copies the custId attribute from this XML data:

<invalidLoanApplication xmlns="http://samples.otn.com">
  <application xmlns = "http://samples.otn.com/XPath/autoloan">
    <customer custId = "111" >
      <name>
        Mike Olive
      </name>
      ...
    </customer>
    ...
  </application>
</invalidLoanApplication>

The following example selects the custId attribute of the customer field and assigns it to the variable custId:

<!-- get the custId attribute and assign to variable custId -->
<assign>
  <copy>
    <from variable="input" part="payload" query="/tns:invalidLoanApplication/autoloan:application/autoloan:customer/@custId"/>
    <to variable="custId"/>
  </copy>
</assign>
Manipulating XML Data with bpelx Extensions

You may want to perform various operations on XML data in assign activities. The following bpelx extension types provide this functionality:

- `bpelx:append`
- `bpelx:insertBefore`
- `bpelx:insertAfter`
- `bpelx:remove`
- `bpelx:rename` and XSD Type Casting
- `bpelx:copyList`

`bpelx:append`

The `bpelx:append` extension in an assign activity enables a BPEL process to append the contents of one variable, expression, or XML fragment to another variable’s contents.

```xml
<bpel:assign>
  <bpelx:append>
    <bpelx:from .../>
    <bpelx:to .../>
  </bpelx:append>
</bpel:assign>
```

The `from-spec` query within `bpelx:append` yields zero or more nodes. The node list is appended as child nodes to the target node specified by the `to-spec` query.

The `to-spec` query must yield one single L-Value element node. Otherwise, a `bpel:selectionFailure` fault is generated. The `to-spec` query cannot refer to a partner link.

The following example consolidates multiple bills of material into one single bill of material by appending multiple `b:part`'s for one BOM to `b:parts` of the consolidated BOM.

```xml
<bpel:assign>
  <bpelx:append>
```

See Also:  The following sample:
- `SOA_Oracle_Home\bpel\samples\references\XPath`
Manipulating XML Data in BPEL

The `<bpelx:insertBefore>` extension in an assign activity enables a BPEL process to insert the contents of one variable, expression, or XML fragment before another variable's contents.

```xml
<bpel:assign>
  <bpelx:insertBefore>
    <bpelx:from variable="billOfMaterialVar" query="/b:bom/b:parts/b:part" />
    <bpelx:to variable="consolidatedBillOfMaterialVar" query="/b:bom/b:parts" />
  </bpelx:insertBefore>
</bpel:assign>
```

The `from-spec` query within `<bpelx:insertBefore>` yields zero or more nodes. The node list is appended as child nodes to the target node specified by the `to-spec` query.

The `to-spec` query of the `insertBefore` operation points to one or more single L-Value nodes. If more than one node is returned, the first node is used as the reference node. The reference node must be an element node. The parent of the reference node must also be an element node. Otherwise, a `bpel:selectionFailure` fault is generated. The node list generated by the `from-spec` query selection is inserted before the reference node. The `to-spec` query cannot refer to a partner link.

The following example shows the syntax before the execution of `<insertBefore>`. The value of `addrVar` is:

```xml
<a:usAddress>
  <a:state>CA</a:state>
  <a:zipcode>94065</a:zipcode>
</a:usAddress>
```

After the execution of the following syntax in the BPEL process file:

```xml
<bpel:assign>
  <bpelx:insertBefore>
    <bpelx:from>
      <a:city>Redwood Shore</a:city>
    </bpelx:from>
    <bpelx:to "addrVar" query="/a:usAddress/a:state" />
  </bpelx:insertBefore>
</bpel:assign>
```

The value of `addrVar` now becomes:

```xml
<a:usAddress>
  <a:city>Redwood Shore</a:city>
  <a:state>CA</a:state>
  <a:zipcode>94065</a:zipcode>
</a:usAddress>
```
Manipulating XML Data with bpelx Extensions

bpelx:insertAfter

The `bpelx:insertAfter` extension in an assign activity enables a BPEL process to insert the contents of one variable, expression, or XML fragment after another variable's contents.

```xml
<bpel:assign>
  <bpel:insertAfter>
    <bpelx:from ... />
    <bpelx:to ... />
  </bpelx:insertAfter>
</bpel:assign>
```

This operation is similar to the functionality described for "bpelx:insertBefore" on page 3-11, except for the following:

- If multiple L-Value nodes are returned by the `to-spec` query, the last node is used as the reference node.
- Instead of inserting nodes before the reference node, the source nodes are inserted after the reference node.

This operation can also be considered a macro of `conditional-switch + (append or insertBefore)`.

The following example shows the syntax before the execution of `<insertAfter>`.

The value of `addrVar` is:

```xml
<a:usAddress>
  <a:addressLine>500 Oracle Parkway</a:addressLine>
  <a:state>CA</a:state>
  <a:zipcode>94065</a:zipcode>
</a:usAddress>
```

After execution of the following syntax in the BPEL process file:

```xml
<bpel:assign>
  <bpel:insertAfter>
    <bpelx:from>
      <a:addressLine>Mailstop 1op6</a:addressLine>
    </bpelx:from>
    <bpelx:to "addrVar" query="/a:usAddress/a:addressLine[1]" />
  </bpelx:insertAfter>
</bpel:assign>
```

The value of `addrVar` becomes:

```xml
<a:usAddress>
  <a:addressLine>500 Oracle Parkway</a:addressLine>
  <a:addressLine>Mailstop 1op6</a:addressLine>
  <a:state>CA</a:state>
  <a:zipcode>94065</a:zipcode>
</a:usAddress>
```

The `from-spec` query within `bpelx:insertAfter` yields zero or more nodes. The node list is appended as child nodes to the target node specified by the `to-spec` query.

bpelx:remove

The `bpelx:remove` extension in an assign activity enables a BPEL process to remove a variable.
Manipulating XML Data with bpelx Extensions

Node removal specified by the XPath expression is supported. Nodes specified by the XPath expression can be multiple, but must be L-Values. Nodes being removed from this parent can be text nodes, attribute nodes, and element nodes.

The XPath expression can return one or more nodes. If the XPath expression returns zero nodes, then a `bpel:selectionFailure` fault is generated.

The syntax of `bpelx:target` is similar to and a subset of `to-spec` for the `copy` operation.

The following example shows `addrVar` with the following value:

```
<a:usAddress>
  <a:addressLine>500 Oracle Parkway</a:addressLine>
  <a:addressLine>Mailstop 1op6</a:addressLine>
  <a:state>CA</a:state>
  <a:zipcode>94065</a:zipcode>
</a:usAddress>
```

After executing the following syntax in the BPEL process file, the second address line of Mailstop is removed:

```
<bpel:assign>
  <bpelx:remove>
    <target variable="addrVar" query="/a:usAddress/a:addressLine[2]" />
  </bpelx:remove>
</bpel:assign>
```

After executing the following syntax in the BPEL process file, both address lines are removed:

```
<bpel:assign>
  <bpelx:remove>
    <target variable="addrVar" query="/a:usAddress/a:addressLine[1]" />
  </bpelx:remove>
</bpel:assign>
```

**bpelx:rename and XSD Type Casting**

The `bpelx:rename` extension in an assign activity enables a BPEL process to rename an element through use of XSD type casting.

```
<bpel:assign>
  <bpelx:rename elementTo="QName1" typeCastTo="QName2">
    <bpelx:target variable="ncname" part="ncname" query="xpath_str" />
  </bpelx:rename>
</bpel:assign>
```

The syntax of `bpelx:target` is similar to and a subset of `to-spec` for the `copy` operation. The target must return a list of one more element nodes. Otherwise, a `bpel:selectionFailure` fault is generated. The element nodes specified in the `from-spec` are renamed the `QName` specified by the `elementTo` attribute. The
xsi:type attribute is added to those element nodes to cast those elements to the QName type specified by the typeCastTo attribute.

Assume you have the following employee list:

```xml
<e:empList>
  <e:emp>
    <e:firstName>John</e:firstName><e:lastName>Dole</e:lastName>
  </e:emp>
  <e:emp>
    <e:firstName>Jane</e:firstName><e:lastName>Dole</e:lastName>
    <e:approvalLimit>3000</e:approvalLimit>
    <e:managing />
  </e:emp>
  <e:emp>
    <e:firstName>Peter</e:firstName><e:lastName>Smith</e:lastName>
  </e:emp>
  <e:emp>
    <e:firstName>Mary</e:firstName><e:lastName>Smith</e:lastName>
  </e:emp>
</e:empList>
```

Promotion changes are now applied to Peter Smith in the employee list:

```xml
<bpel:assign>
  <bpelx:rename typeCastTo="e:ManagerType">
    <bpelx:target variable="empListVar" query="/e:empList/e:emp[./e:firstName='Peter' and ./e:lastName='Smith']" />
  </bpelx:rename>
</bpel:assign>
```

After executing the above casting (renaming), the data looks as follows with xsi:type info added to Peter Smith:

```xml
<e:empList>
  <e:emp>
    <e:firstName>John</e:firstName><e:lastName>Dole</e:lastName>
  </e:emp>
  <e:emp>
    <e:firstName>Jane</e:firstName><e:lastName>Dole</e:lastName>
    <e:approvalLimit>3000</e:approvalLimit>
    <e:managing />
  </e:emp>
  <e:emp>
    <e:firstName>Peter</e:firstName><e:lastName>Smith</e:lastName>
    xsi:type="e:ManagerType"
  </e:emp>
  <e:emp>
    <e:firstName>Mary</e:lastName>Smith</e:lastName>
  </e:emp>
</e:empList>
```

The employee data of Peter Smith is now invalid, because <approvalLimit> and <managing> are missing. Therefore, <append> is used to add that information.

```xml
<bpel:assign>
  <bpelx:rename typeCastTo="e:ManagerType">
    <bpelx:target variable="empListVar" query="/e:empList/e:emp[./e:firstName='Peter' and ./e:lastName='Smith']" />
  </bpelx:rename>
  <bpelx:append>
    <e:approvalLimit>3000</e:approvalLimit>
    <e:managing />
  </bpelx:append>
</bpel:assign>
```
Manipulating XML Data in BPEL

```xml
<bpelix:from>
    <e:approvalLimit>2500</e:approvalLimit>
    <e:managing />
</bpelix:from>

<bpelix:to variable="empListVar"
    query="/e:empList/e:emp[./e:firstName='Peter' and
    ./e:lastName='Smith']" />
</bpelix:assign>
```

With the execution of both `rename` and `append`, the corresponding data looks as follows:

```xml
<e:emp xsi:type="e:ManagerType">
    <e:firstName>Peter</e:firstName>
    <e:lastName>Smith</e:lastName>
    <e:approvalLimit>2500</e:approvalLimit>
    <e:managing />
</e:emp>
```

### bpelix:copyList

The `bpelix:copyList` extension in an assign activity enables a BPEL process to perform a `copyList` operation of the contents of one variable, expression, or XML fragment to another variable.

```xml
<bpelix:assign>
    <bpelix:copyList>
        <bpelix:from ... />   
        <bpelix:to ... /> 
    </bpelix:copyList>
</bpelix:assign>
```

The `from-spec` query can yield a list of either all attribute nodes or all element nodes. The `to-spec` query can yield a list of L-value nodes — either all attribute nodes or all element nodes.

All the element nodes returned by the `to-spec` query must have the same parent element. If the `to-spec` query returns a list of element nodes, all element nodes must be contiguous.

If the `from-spec` query returns attribute nodes, then the `to-spec` query must return attribute nodes. Likewise, if the `from-spec` query returns element nodes, then the `to-spec` query must return element nodes. Otherwise, a `bpws:mismatchedAssignmentFailure` fault is thrown.

The `from-spec` query can return zero nodes, while the `to-spec` query must return at least one node. If the `from-spec` query returns zero nodes, the effect of the `copyList` operation is similar to the `remove` operation.

The `copyList` operation provides the following features:

- Removes all the nodes pointed to by the `to-spec` query
- If the `to-spec` query returns a list of element nodes and there are leftover child nodes after removal of those nodes, the nodes returned by the `from-spec` query are inserted before the next sibling of the last element specified by the `to-spec` query. If there are no leftover child nodes, an `append` operation is performed.
- If the `to-spec` query returns a list of attribute nodes, those attributes are removed from the parent element. Then, the attributes returned by the `from-spec` query are appended to the parent element.
Validating XML Data with `bpelx:validate`

The `bpelx:validate` function enables you to verify code and identify invalid XML data. Use this extension as follows:

- With the `validate` attribute in an `assign` activity:
  ```xml
  <assign bpelx:validate="yes|no">...
  </assign>
  ```

- In `<bpelx:validate>` as a standalone extended activity that can be used without an `assign` activity:
  ```xml
  <bpelx:validate variables="NCNAMES" />
  
  For example:
  ```xml
  <bpelx:validate variables="myMsgVariable myPOElemVar" />
  ```

If you want to verify the validity of XML data, set the `validateXML` property to `true` in the Manage BPEL Domain window of Oracle BPEL Control.

Manipulating XML Data Sequences That Use Arrays

Data sequences are one of the most basic data models used in XML. However, manipulating them can be nontrivial. One of the most common data sequence patterns used in BPEL processes are arrays. Based on the XML schema, the way you can identify a data sequence definition is by its attribute `maxOccurs` being set to a value of more than one or marked as unbounded. See the XML Schema Specification at [http://www.w3.org/TR](http://www.w3.org/TR) for more information.

The examples in this section illustrate several basic ways of manipulating data sequences in BPEL. However, there are other associated requirements, such as performing looping or dynamic referencing of endpoints. For additional code samples and further information regarding real-world use cases for data sequence manipulation in BPEL, see [http://www.oracle.com/technology/bpel](http://www.oracle.com/technology/bpel).

Each of the following sections describes a particular requirement for data sequence manipulation. For a code example that describes all data sequences, see `ArraySample.bpel`, which takes a data sequence as input and loops through it, adding together individual line items in each data sequence element into a total value.

**See Also:** The `ArraySample.bpel` sample file located at:

- `SOA_Oracle_Home/bpel/samples/tutorial/112.Arrays`

Statically Indexing into an XML Data Sequence That Uses Arrays

The following two examples illustrate how to use XPath functionality to select a data sequence element when the index of the element you want is known at design time. In these cases, it is the first element.

In the following example, `addresses[1]` selects the first element of the addresses data sequence:

```xml
<assign>
  <!-- get the first address and assign to variable address -->
  <copy>
    <from variable="input" part="payload" query="/tns:invalidLoanApplication/autoloan:application
    
  </copy>
</assign>
```

3-16  Oracle BPEL Process Manager Developer's Guide
In this query, addresses[1] is equivalent to addresses[position()=1], where position is one of the core XPath functions (see sections 2.4 and 4.1 of the XML Path Language (XPath) Specification). The query in the next example calls the position function explicitly to select the first element of the addresses data sequence. It then selects that address’s street element (which the activity assigns to the variable street1).

If you review the definition of the input variable and its payload part in the WSDL file, you go several levels down before coming to the definition of the addresses field. There you see the maxOccurs="unbounded" attribute. The two XPath indexing methods are functionally identical; you can use whichever method you prefer.

Determining Sequence Size

If you need to know the run-time size of a data sequence—that is, the number of nodes or data items in the sequence—you can get it by using the combination of the XPath built-in count() function and the BPEL built-in getVariableData() function. This example calculates the number of elements in the item sequence and assigns it to the integer variable lineItemSize:

See Also: The following sample:
- SOA_Oracle_Home\bpel\samples\references\XPath

Dynamically Indexing by Applying a Trailing XPath to an Expression

Often a dynamic value is needed to index into a data sequence—that is, you need to get the nth node out of a sequence, where the value of n is defined at run time. This section covers the following methods for dynamically indexing by applying a trailing XPath into expressions:

See Also: The following sample:
- SOA_Oracle_Home\bpel\samples\references\XPathFunction
Manipulating XML Data Sequences That Use Arrays

- Dynamic Indexing Example
- Using the bpelx:append Extension to Append New Items to a Sequence
- Merging Data Sequences
- Dynamically Indexing with the BPEL getElement Function
- Generating Functionality Equivalent to an Array of an Empty Element

Dynamic Indexing Example

The dynamic indexing method shown here applies a trailing XPath to the result of bpws:getVariableData(), instead of using an XPath as the last argument of bpws:getVariableData(). The trailing XPath references to an integer-based index variable within the position predicate (that is, [ ... ]):

```xml
<variable name="idx" type="xsd:integer"/>
```

```xml
<assign>
  <copy>
    <from expression="bpws:getVariableData('input','payload' /p:line-item[bpws:getVariableData('idx')]/p:line-total" />
    <to variable="lineTotalVar" />
  </copy>
</assign>
```

Assume at run time that the `idx` integer variable holds `2` as its value. The preceding expression within the `from` is equivalent to:

```xml
<from expression="bpws:getVariableData('input','payload' /p:line-item[2]/p:line-total" />
```

There are some subtle XPath usage differences, when an XPath used trailing behind the bpws:getVariableData() function is compared with the one used inside the function.

Using the same example (where `payload` is the message part of element "p:invoice"), if the XPath is used within the getVariableData() function, the root element name ("p:invoice") must be specified at the beginning of the XPath. For example:

```xml
bpws:getVariableData('input','payload' /p:invoice/p:line-item[2]/p:line-total')
```

If the XPath is used trailing behind the bpws:getVariableData() function, the root element name does not need to be specified in the XPath. For example:

```xml
bpws:getVariableData('input','payload')/p:line-item[2]/p:line-total
```

This is because the node returned by the getVariableData() function is already the root element. Specifying the root element name again in the XPath is redundant and is standard XPath semantics.

Using the bpelx:append Extension to Append New Items to a Sequence

The bpelx:append extension in an assign activity enables BPEL processes to append new elements to an existing parent element:

```xml
<assign name="assign-3">
  <copy>
```

3-18  Oracle BPEL Process Manager Developer's Guide
Manipulating XML Data Sequences That Use Arrays

The `bpelx:append` logic in this example appends the payload element of the `partInfoResultVar` variable as a child to the payload element of the `output` variable. In other words, the payload element of `output` variable is used as the parent element.

**See Also:** The following samples:
- "Assign Activity" on page B-3 for details about using multiple copies of this extension in a single assign activity
- SOA_Oracle_Home\bpel\samples\tutorials\126.DataAggregator\AggregationTutorial

**Merging Data Sequences**

You can merge two sequences into a single data sequence. This pattern is common when the data sequences are in an array (that is, the sequence of data items of compatible types).

The following two `append` operations under `assign` demonstrate how to merge data sequences:

```xml
<assign>
  <!-- initialize "mergedLineItems" variable to an empty element -->
  <copy>
    <from> <p:lineItems /> </from>
    <to variable="mergedLineItems" />
  </copy>
  <bpelx:append>
    <bpelx:from variable="input" part="payload" query="/p:invoice/p:lineItems/p:lineitem" />
    <bpelx:to variable="mergedLineItems" />
  </bpelx:append>
  <bpelx:append>
    <bpelx:from variable="literalLineItems" query="/p:lineItems/p:lineitem" />
    <bpelx:to variable="mergedLineItems" />
  </bpelx:append>
</assign>
```

**See Also:** The `ArraySample.bpel` sample file located at:
- SOA_Oracle_Home\bpel\samples\tutorials\112.Arrays

**Dynamically Indexing with the BPEL getElement Function**

If you do not want to use the two-step process of creating an XPath query to dynamically index into a sequence, you can use the XPath function `getElement`
Manipulating XML Data Sequences That Use Arrays

instead. This function takes a sequence and an index (which can be a dynamic value, such as a variable) and returns the appropriate sequence element.

```xml
<variable name="lineItemIndex" type="xsd:int"/>
<!-- execute the XPath extension function getElement(arrayOfElements[], index) to fetch one element from an array of elements -->
<assign>
  <copy>
    <from expression="ora:getElement('output', 'payload', '/invoice/lineItems/item', bpws:getVariableData('lineItemIndex'))"/>
    <to variable="myLineItem"/>
  </copy>
</assign>
```

Note: The XPath function getElement is being deprecated in a future release.

See Also: The following sample:
- SOA_Oracle_Home\bpel\samples\references\XPathFunction

Generating Functionality Equivalent to an Array of an Empty Element

The `genEmptyElem` function generates functionality equivalent to an array of an empty element to an XML structure. This function takes the following arguments:

```
genEmptyElem('ElemQName',int?, 'TypeQName'? , boolean?)
```

Note the following issues:
- The first argument specifies the QName of the empty elements.
- The optional second integer argument specifies the number of empty elements. If missing, the default size is 1.
- The third optional argument specifies the QName, which is the xsi:type of the generated empty name. This xsi:type pattern matches the SOAPENC:Array. If it is missing or is an empty string, the xsi:type attribute is not generated.
- The fourth optional Boolean argument specifies whether the generated empty elements are XSI-nil. If missing or false, xsi:nil is not generated.

The following example shows an append statement initializing a purchase order (PO) document with 10 empty `<lineItem>` elements under `po`:

```xml
<bpelx:assign>
  <bpelx:append>
    <bpelx:from expression="ora:genEmptyElem('p:lineItem',10)" />
    <bpelx:to variable="poVar" query="/p:po" />
  </bpelx:append>
</bpelx:assign>
```

The `genEmptyElem` function in this example can be replaced with an embedded XQuery expression:

```xml
ora:genEmptyElem('p:lineItem',10)
```
Converting from a String to an XML Element

Sometimes a service is defined to return a string, but the content of the string is actually XML data. The problem is that, although BPEL provides support for manipulating XML data (using XPath queries, expressions, and so on), this functionality is not available if the variable or field is of type string. With Java, you use document object model (DOM) functions to convert the string to a structured XML object type. You can use the BPEL XPath function `parseEscapedXML` to do the same thing. This function takes XML data, parses it through DOM, and returns structured XML data that can be assigned to a typed BPEL variable. For example:

```xml
<assign>
  <copy>
    <from expression="ora:parseEscapedXML('<item xmlns="http://samples.otn.com" sku="006">
      <description>sun ultra sparc VI server</description>
      <price>1000</price>
      <quantity>2</quantity>
      <lineTotal>2000</lineTotal>
    </item>')" />
  </copy>
</assign>
```

**See Also:** "genEmptyElem" on page D-41

SOAP-Encoded Arrays Not Supported

Oracle BPEL Process Manager does not support SOAP-encoded arrays (soapenc:arrayType).

Use one of the following workarounds:

- Apache Axis supports document-literal style services. This means you can change the service to not use soapenc:arrayType.
- A wrapper can be placed around the service (also using Apache Axis) so that the BPEL process talks to the document literal wrapper service, which in turn calls the underlying service with soapenc:arrayType.
- Call a service with soapenc:arrayType from BPEL, but construct the XML message more manually in the BPEL code. This enables you to avoid changing or wrapping the service. However, each time you want to call that service from BPEL, you must take extra steps.

Converting from a String to an XML Element

The empty elements generated by this function are typically invalid XML data. You perform further data initialization after the empty elements are created. Using the same example above, you can perform the following:

- Add attribute and child elements to those empty lineItem elements.
- Perform `copy` operations to replace the empty elements. For example, copy from a Web service result to an individual entry in this equivalent array under a flowN activity.

**See Also:** "genEmptyElem" on page D-41
Differences Between Document-Style and RPC-Style WSDL Files

The examples shown up to this point have been for document-style WSDL files, in which a message is defined with an XML schema element, as in the following example:

```xml
<message name="LoanFlowRequestMessage">
    <part name="payload" element="s1:loanApplication"/>
</message>
```

This is in contrast to RPC-style WSDL files, in which the message is defined with an XML schema type, as in:

```xml
<message name="LoanFlowRequestMessage">
    <part name="payload" type="s1:LoanApplicationType"/>
</message>
```

This affects the material in this chapter because there is a difference in how XPath queries are constructed for the two WSDL message styles. For an RPC-style message, the top-level element (and therefore the first node in an XPath query string) is the part name (`payload` in the previous example). In document-style, the top-level node is the element name (for example, `loanApplication`).

The following example shows what an XPath query string looks like if the LoanServices used in BPEL demo applications (such as LoanFlow) were RPC style.

**RPC-Style WSDL File**

```xml
<message name="LoanServiceResultMessage">
    <part name="payload" type="s1:LoanOfferType"/>
</message>
```

```xml
<complexType name="LoanOfferType">
    <sequence>
        <element name="providerName" type="string"/>
        <element name="selected" type="boolean"/>
        <element name="approved" type="boolean"/>
        <element name="APR" type="double"/>
    </sequence>
</complexType>
```

**RPC-Style BPEL File**

```xml
<variable name="output" messageType="tns:LoanServiceResultMessage"/>

<assign>
    <copy>
        <from expression="9.9"/>
        <to variable="output" part="payload" query="/payload/APR"/>
    </copy>
</assign>
```

---

**See Also:** The following sample:

- `SOA_Oracle_HOME\bpe1\samples\references\XPathFunction`
Adding a Custom WSIF Provider

You can add a custom WSIF provider to Oracle BPEL Process Manager by performing the following tasks:

- **Task 1: Register the WSDL Extension to the WSDL Reader**
  1. Create your WSDL extension object model by following the Java WSDL extension included in the Apache WSIF package. An `extensionRegistry` class, such as `JavaExtensionRegistry.java` in the WSIF Java package, is needed to register the WSDL extension.
  2. Create a `SOA_Oracle_Home\bpel\system\classes\WSDLExtensions` file if it does not currently exist.
  3. Add the name of the class on its own line in the file.

This file is picked up by the WSDL reader and enables your custom WSDL extension to be recognized when the WSDL is parsed.

- **Task 2: Register the WSIF Provider**
  1. Develop your custom WSIF provider by following the example of any providers included with the Apache WSIF package.
  2. Modify the `wsif-providers` entry in the `SOA_Oracle_Home\bpel\system\config\collaxa-config.xml` file as follows:

```xml
<property id="wsif-providers">
  <name>Custom WSIF providers for Oracle BPEL server</name>
  <value>provider-class-name</value>
  <comment><![CDATA[The value should be a comma separated list of WSIF provider class names.]]></comment>
</property>
```

4. Go to Oracle BPEL Admin Console:
   
   `http://localhost:port/BPELAdmin/server.jsp`

See Also:
- The following samples:
  - `SOA_Oracle_Home\bpel\samples\utils\AsyncLoanService` (LoanServices)
  - `SOA_Oracle_Home\bpel\samples\demos\LoanDemo\LoanFlow` (BPEL demo application)
5. Log in as oc4jadmin/password when prompted.
6. Click the WSIF tab.

The loaded WSIF providers display in a table.

### Input and Output Message Header Handling

Oracle BPEL Process Manager uses WSIF to call Web services. The WSIF API is simple to invoke. When you invoke a Web service operation, you perform `operation.invoke(input, output)`.

However, input and output messages sometimes do not contain all the information that is transferred over the transport layer. In this case, the WSIF API appears oversimplified. However, WSIF provides a `MessageContext` that you can set to the operation to perform. You can bind extra information (that is, SOAP headers and HTTP headers) to the context; the WSIF provider can send this information to the service.

#### Header Handlers

The following header handlers in the Web services invocation layer of Oracle BPEL Process Manager are provided.

- An `InputHeaderHandler` is invoked before calling the WSIF provider.
- An `OutputHeaderHandler` is invoked after calling the WSIF provider.

Figure 3-1 shows this functionality.

```java
public interface HeaderHandler {
   invoke(CXPartnerLink partnerLink, 
          String operationName, 
          Map payload, 
          Map headers, 
          Map callProps);
}
```

The payload is the input or output message. The header is a map that contains messages that are used as SOAP headers.
The partnerLink is the object model. The input header handler can get information from the partnerLink and set it to the context. The output header handler can extract information from the context and bind it to the partnerLink.

The following example shows the use of a RequestHeaderHandler:

```java
public class MyTestRequestHeaderHandler {
    public void invoke(CXPartnerLink partnerLink, String operationName, 
                    Map payload, Map header, Map callProps) {
        System.out.println("in MyTestRequestHeaderHandler ...!");
        Map httpHeaders = (Map) callProps.get("http-request-headers");
        if (httpHeaders == null) {
            httpHeaders = new HashMap();
            callProps.put("httpRequestHeaders", httpHeaders);
        }
        httpHeaders.put("myHeader1", partnerLink.getProperty("myHeader1"));
        httpHeaders.put("myHeader2", partnerLink.getProperty("myHeader2"));
    }
}
```

The header information bound to the context is still abstract in that it is transport agnostic. The underlying WSIF provider must recognize this information from the context and set and get it correctly in the transport layer.

Registering of Header Handlers

The registration of the header handlers is performed in the bpel.xml deployment descriptor file:

```xml
<BPELSuitcase>
    <BPELProcess src="QuoteConsumer.bpel" id="QuoteConsumer">
        <partnerLinkBindings>
            <partnerLinkBinding name="client">
                <property name="wsdlLocation">QuoteConsumer.wsdl</property>
            </partnerLinkBinding>
            <partnerLinkBinding name="StockQuoteService">
                <property name="wsdlLocation">http://glennmi:9700/orabpel/default/StockQuoteService/StockQuoteService?wsdl</property>
                <property name="requestHeaderHandlers">my.custom.MyRequestHeaderHandler</property>
                <property name="responseHeaderHandlers">my.custom.MyResponseHeaderHandler</property>
            </partnerLinkBinding>
        </partnerLinkBindings>
        <configurations>
            <!-- Optional property used to customize the BPEL console test form. -->
            <property name="testIntroduction">This sample shows how to use the BPEL invoke activity to invoke a synchronous stock quote service.</property>
        </configurations>
    </BPELProcess>
</BPELSuitcase>
```
Manipulation of partnerLink Properties

The headers are bound to the partnerLink. Some use cases require partnerLink properties to be retrieved and set in the BPEL process. The partnerLink properties are not described in the Business Process Execution Language for Web Services Specification, but an extension can perform this task:

```xml
<process>
  <assign>
    <copy>
      <from partnerLink="p1" bpelx:property="sessionId"/>
      <to partnerLink="p2" bpelx:property="sessionId"/>
    </copy>
    <copy>
      <from variable="var"/>
      <to partnerLink="p2" bpelx:property="sessionId"/>
    </copy>
  </assign>
</BPELProcess>
```

Manipulating SOAP Headers in BPEL

BPEL’s communication activities (invoke, receive, reply, and onMessage) receive and send messages through specified message variables. These default activities permit one variable to operate in each direction. For example, the invoke activity has inputVariable and outputVariable attributes. You can specify one variable for each of the two attributes. This is enough if the particular operation involved uses only one payload message in each direction.

However, WSDL supports more than one message in an operation. In the case of SOAP, multiple messages can be sent along the main payload message as SOAP headers. However, BPEL’s default communication activities cannot accommodate the additional header messages.

Oracle BPEL Process Manager solves this problem by extending the default BPEL communication activities with the bpelx:headerVariable extension. The extension syntax is as follows:

```xml
<invoke bpelx:inputHeaderVariable="inHeader1 inHeader2 ...">
  bpelx:outputHeaderVariable="outHeader1 outHeader2 ...">
    ...
</invoke>

<receive bpelx:headerVariable="inHeader1 inHeader2 ...">
  <onMessage bpelx:headerVariable="inHeader1 inHeader2 ...">
    ...
  </onMessage>
  <reply bpelx:headerVariable="inHeader1 inHeader2 ...">
    ...
  </reply>
</receive>
```

Receiving SOAP Headers in BPEL

This section provides an example of how to create BPEL and WSDL files to receive SOAP headers.

1. Create a WSDL file that declares header messages and the SOAP binding that binds them to the SOAP request.

```xml
<message name="MessageIDHeader">
  <part name="MessageID" element="wsa:MessageID"/>
</message>
<message name="ReplyToHeader">
  <part name="ReplyTo" element="wsa:ReplyTo"/>
</message>
```
Manipulating SOAP Headers in BPEL

[Code]

<!-- custom header -->
<message name="CustomHeaderMessage">
  <part name="header1" element="tns:header1"/>
  <part name="header2" element="tns:header2"/>
</message>

<binding name="HeaderServiceBinding" type="tns:HeaderService">
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="initiate">
    <soap:operation style="document" soapAction="Initiate"/>
    <input>
      <soap:header message="tns:ReplyToHeader" part="ReplyTo" use="literal"/>
      <soap:header message="tns:MessageIDHeader" part="MessageID" use="literal"/>
      <soap:header message="tns:CustomHeaderMessage" part="header1" use="literal"/>
      <soap:header message="tns:CustomHeaderMessage" part="header2" use="literal"/>
      <soap:body use="literal"/>
    </input>
  </operation>
</binding>

2. Create a BPEL source file that declares the header message variables and uses bpelx:headerVariable to receive the headers.

<variables>
  <variable name="input" messageType="tns:HeaderServiceRequestMessage"/>
  <variable name="event" messageType="tns:HeaderServiceEventMessage"/>
  <variable name="output" messageType="tns:HeaderServiceResultMessage"/>
  <variable name="customHeader" messageType="tns:CustomHeaderMessage"/>
  <variable name="messageID" messageType="tns:MessageIDHeader"/>
  <variable name="replyTo" messageType="tns:ReplyToHeader"/>
</variables>

<sequence>
  <!-- receive input from requestor -->
  <receive name="receiveInput" partnerLink="client" portType="tns:HeaderService" operation="initiate" variable="input" bpelx:headerVariable="customHeader messageID replyTo" createInstance="yes"/>
</sequence>

Sending SOAP Headers in BPEL

This section provides an example of how to send SOAP headers.

1. Define the partnerLinkBinding definition in bpel.xml to refer to the HeaderService WSDL.

2. Define the custom header variable, manipulate it, and send it using bpelx:inputHeaderVariable.

<variables>
Using Binary Attachments in SOAP Messages

There are two supported methods for transferring opaque data in a SOAP call:

- **Embedding data**
  
  This method embeds opaque data as element or attribute content. XML supports opaque data as content through either base64 or hexadecimal text encoding. XML schema’s two binary data types, `xs:base64Binary` and `xs:hexBinary`, are used with this method. Since the opaque data is converted to a basic XML schema type, it can be manipulated like other XML data in a BPEL process in terms of being assigned, sent, and received in standard BPEL activities.

  Data encoded in base64 format expands by a factor of 1.33 times the original size. Hexadecimal encoded data expands by a factor of 2 times. This is assuming an underlying UTF-8 text encoding is used in both cases; if the underlying text encoding used is UTF-16, these numbers double. To achieve better performance at both the SOAP layer and BPEL execution layer, keep the original data as it is when transferring it through SOAP. The second method described below, SOAP with an attachment, provides a technique for doing this.

- **SOAP with an attachment**
  
  SOAP with an attachment can be achieved through use of HTTP with the Multipurpose Internet Mail Extensions (MIME) or Direct Internet Message Encapsulation (DIME) protocols. The SOAP envelope and opaque data are wrapped in MIME or DIME sections.

  The following example shows a message using SOAP with an attachment through use of the MIME protocol. The attachment is passed by reference through use of an identifying key, instead of being copied. Note that with MIME, the XML part and the binary part must appear in separate parts. The XML part contains a reference to the binary attachment part. This differs from DIME, which does not have this restriction.

```
MIME-Version: 1.0
Content-Type: multipart/Related; boundary=MIME_boundary;
type=text/xml; start="<mymessage.xml@collaxa.com>
Content-Description: A SOAP Envelope with pdf

-->MIME_boundary
Content-Type: text/xml; charset=UTF-8
Content-Transfer-Encoding: 8bit
Content-ID: <mymessage.xml@collaxa.com>
```
Using Binary Attachments in SOAP Messages

<s:Envelope xmlns:s='http://www.w3.org/2002/12/soap-envelope' >
  <s:Body>
    <m:applyLoan xmlns:m='http://samples.Collaxa.com/MIMEService' >
      <customerName>John Doe</customerName>
      <pdf data="1234567890" />
    </m:applyLoan>
  </s:Body>
</s:Envelope>

--MIME_boundary
Content-Type: application/pdf
Content-Transfer-Encoding: binary
Content-Location: 1234567890
fd a5 8a 29 aa 46 1b 24
--MIME_boundary

The binary data is still typed as base64binary or hexBinary. However, the data remains in binary format through transporting and processing. There is no overhead added with encoding and decoding.

---

**Note**: If you use large binary attachment files in SOAP messages with Oracle Database Lite, your BPEL process may not complete processing, which can cause you to run out of system memory. Oracle Database Lite is largely for testing purposes. To use large binary attachment files in SOAP messages, use an Oracle Database as your dehydration store.

---

**See Also**:
- SOA_Oracle_Home\bpel\samples\demos\Attachment for demonstrations of using SOAP with an attachment through MIME and DIME

**Use Case: SOAP Message with Binary Attachment Using MIME**

This section provides a use case and describes design implementation issues. You cannot currently model a SOAP message with an attachment through Oracle JDeveloper. You must manually edit the necessary BPEL and WSDL files.

In this use case:
- The BPEL process acts as a service to receive and reply to a SOAP message with an attachment.
- The BPEL process acts as a client to send and receive a response to a SOAP message with an attachment.
- Binary data is assigned to another variable.
- Binary data is read from a URL.
- Binary data is saved to a local file.

Two BPEL processes are constructed:
- MIMEService is essentially an echo service of a SOAP message with an attachment. This process does the following:
  - Receives a SOAP message with an attachment and saves the opaque data to a local file.
Assigns the opaque data to an output variable.

- Uses the output variable to reply to a SOAP message with an attachment to the invoker.

- MIMERequester is the client of the first service. This process does the following:
  - Reads the binary data from a URL and assigns it to a variable.
  - Uses the variable as input to perform an invoke on the first process.
  - Receives a SOAP message with an attachment.

**WSDL File Contents**
The following WSDL file defines MIMEService. This file uses the WSDL binding MIME extension to define the SOAP message with an attachment.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<definitions>
  ...
  ...
  <schema xmlns="http://www.w3.org/2001/XMLSchema">
  </schema>
  ...
  ...
  <message name="ReplyToHeader">
    <part name="ReplyTo" element="wsa:ReplyTo" />
  </message>
  <message name="MessageIDHeader">
    <part name="MessageID" element="wsa:MessageID" />
  </message>
  <message name="RelatesToHeader">
    <part name="RelatesTo" element="wsa:RelatesTo" />
  </message>
  <message name="MIMEServiceRequestMessage">
    <part name="payload" type="xsd:int"/>
    <part name="bin" type="xsd:anyType"/>
  </message>
  <message name="MIMEServiceResponseMessage">
    <part name="payload" type="xsd:int"/>
    <part name="bin" type="xsd:anyType"/>
  </message>
  <portType name="MIMEService">
    <operation name="initiate">
      <input message="tns:MIMEServiceRequestMessage"/>
    </operation>
    <operation name="process">
      <input message="tns:MIMEServiceRequestMessage"/>
      <output message="tns:MIMEServiceResponseMessage"/>
    </operation>
  </portType>
  <portType name="MIMEServiceCallback">
    <operation name="onResult">
      <input message="tns:MIMEServiceResponseMessage"/>
    </operation>
  </portType>
</definitions>
```
<binding name="MIMEServiceBinding" type="tns:MIMEService">
  <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="initiate">
    <soap:operation style="rpc" soapAction="initiate"/>
    <input>
      <mime:multipartRelated>
        <mime:part>
          <soap:header message="tns:MessageIDHeader" part="MessageID" use="literal"/>
          <soap:header message="tns:ReplyToHeader" part="ReplyTo" use="literal"/>
          <soap:body parts="payload" use="literal"/>
        </mime:part>
        <mime:part>
          <mime:content part="bin" type="binary"/>
        </mime:part>
      </mime:multipartRelated>
    </input>
  </operation>
  <operation name="process">
    <soap:operation style="rpc" soapAction="process"/>
    <input>
      <mime:multipartRelated>
        <mime:part>
          <soap:header message="tns:RelatedToHeader" part="RelatedTo" use="literal"/>
          <soap:body parts="payload" use="literal"/>
        </mime:part>
        <mime:part>
          <mime:content part="bin" type="binary"/>
        </mime:part>
      </mime:multipartRelated>
    </input>
    <output>
      <mime:multipartRelated>
        <mime:part>
          <soap:body parts="payload" use="literal"/>
        </mime:part>
        <mime:part>
          <mime:content part="bin" type="binary"/>
        </mime:part>
      </mime:multipartRelated>
    </output>
  </operation>
</binding>

<binding name="MIMEServiceCallbackBinding" type="tns:MIMEServiceCallback">
  <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="onResult">
    <soap:operation style="rpc" soapAction="onResult"/>
    <input>
      <mime:multipartRelated>
        <mime:part>
          <soap:header message="tns:RelatedToHeader" part="RelatedTo" use="literal"/>
          <soap:body use="literal"/>
        </mime:part>
      </mime:multipartRelated>
    </input>
  </operation>
</binding>
Using Binary Attachments in SOAP Messages

```xml
<mime:part>
  <mime:content part="bin" type="binary"/>
</mime:part>
</mime:multipartRelated>
</input>
</operation>
</binding>

<plnk:partnerLinkType name="MIMEService">
  <plnk:role name="MIMEServiceProvider">
    <plnk:portType name="tns:MIMEService"/>
  </plnk:role>
  <plnk:role name="MIMEServiceRequester">
    <plnk:portType name="tns:MIMEServiceCallback"/>
  </plnk:role>
</plnk:partnerLinkType>
</definitions>

In this WSDL, the schema type of the binary data is `xsd:anyType`. This ensures that payload validation is successful when using `xsd:anyType` with either MIME or DIME data.

BPEL File Contents

The `MIMERequester.bpel` file shows how to use binary data in BPEL:

```bpel
<process . . .
  . . .
  . . .
  <sequence>
    <!-- receive input from requester -->
    <receive name="receiveInput" partnerLink="client"
      portType="tns:MIMERequester"
      operation="initiate" variable="input"
      createInstance="yes"/>
    <!-- initialize the input of MIMEService -->
    <assign>
      <copy>
        <from variable="input" part="payload" query="/tns:value"/>
        <to variable="request" part="payload" query="/payload"/>
      </copy>
      <copy>
        <from expression="ora:readBinaryFromFile('request.bin')"/>
        <to variable="request" part="bin"/>
      </copy>
    </assign>
    <invoke name="invoke" partnerLink="MIMEService"
      portType="services:MIMEService"
      operation="initiate" inputVariable="request"/>
    <receive name="receive" partnerLink="MIMEServiceCallback"
      portType="services:MIMEServiceCallback"
      operation="onResult" variable="response"/>
    <assign>
      <copy>
        <from variable="response" part="payload" query="/payload"/>
        <to variable="request" part="payload" query="/payload"/>
      </copy>
    </assign>
    <invoke name="invoke" partnerLink="MIMEService"
      portType="services:MIMEService"
      operation="initiate" inputVariable="request"/>
  </sequence>
</process>
```

In this WSDL, the schema type of the binary data is `xsd:anyType`. This ensures that payload validation is successful when using `xsd:anyType` with either MIME or DIME data.
Using Binary Attachments in SOAP Messages

In this example, XPath extension function `ora:readBinaryFromFile()` reads the binary file and `ora:writeBinaryToFile()` writes the binary content to a file. The binary data can be assigned to another variable like a normal XML document by using the standard BPEL assign activity. The BPEL assign activity is extended here to accommodate the binary data.

Java Client Using SAAJ

MIMEService can be accessed from a Java client. There are two access options:

- Java API for XML-Based RPC (JAX-RPC)
- SOAP with Attachments API Java (SAAJ)

Example 3–1 uses Axis' implementation of SAAJ to invoke MIMEService. This example is used to unit test the interoperability of the created service. The sample request sent by this example is shown in Example 3–2 on page 3-35.

Example 3–1 SAAJ Example

```java
public boolean initiateUsingSAAJ(String filename) throws Exception {
    String endPointURLString =  "http://localhost:" + opts.getPort() +
    "/orabpel/default/MIMEService/1.0";
    SOAPConnectionFactory soapConnectionFactory =
    javax.xml.soap.SOAPConnectionFactory.newInstance();
    SOAPConnection soapConnection =
    soapConnectionFactory.createConnection();
    MessageFactory messageFactory =
```
Using Binary Attachments in SOAP Messages

```java
MessageFactory.newInstance();
SOAPMessage soapMessage =
    messageFactory.createMessage();
MimeHeaders hd = soapMessage.getMimeHeaders();
hd.addHeader("SOAPAction", "initiate");
SOAPPart soapPart = soapMessage.getSOAPPart();
SOAPEnvelope requestEnvelope =
    soapPart.getEnvelope();
SOAPBody body = requestEnvelope.getBody();
SOAPBodyElement operation = body.addChildElement(
    requestEnvelope.createName("initiate");
Vector dataHandlersToAdd = new Vector();
dataHandlersToAdd.add(new DataHandler(new FileDataSource(new
    File(filename))));
javax.xml.soap.SOAPElement element1 =
    operation.addChildElement(requestEnvelope.createName("payload"));
    element1.addAttribute("href", "cid:" + attachment.getContentId());
if (dataHandlersToAdd != null)
    ListIterator dataHandlerIterator =
        dataHandlersToAdd.listIterator();
    while (dataHandlerIterator.hasNext()) {
        DataHandler dataHandler = (DataHandler)
            dataHandlerIterator.next();
        javax.xml.soap.SOAPElement element =
            operation.addChildElement(requestEnvelope.createName("bin"));
        javax.xml.soap.AttachmentPart attachment = 
            soapMessage.createAttachmentPart(dataHandler);
        soapMessage.addAttachmentPart(attachment);
        element.addAttribute(requestEnvelope.createName("href"),
            "cid:" + attachment.getContentId());
    }
javax.xml.soap.SOAPMessage returnedSOAPMessage = 
    soapConnection.call(soapMessage, endPointURLString);
if (returnedSOAPMessage == null)
    return true;
Iterator iterator = returnedSOAPMessage.getAttachments();
    if (!iterator.hasNext()) {
        //The wrong type of object that what was expected.
        System.out.println("Received problem response from server");
        throw new AxisFault("", "Received problem response from server", null, null);
    }
    //Still here, so far so good.
    //Now lets brute force compare the source attachment
    // to the one we received.
    DataHandler rdh = (DataHandler)
        ((AttachmentPart)iterator.next()).getDataHandler();
    //From here we'll just treat the data resource as file.
    String receivedfileName = rdh.getName(); //Get the filename.
    if (receivedfileName == null) {
```

3-34 Oracle BPEL Process Manager Developer's Guide
Example 3–2 shows the sample request sent by the SAAJ program:

Example 3–2 Sample Request

```java
POST /orabpel/default/MIMEService/1.0 HTTP/1.0
Content-Type: multipart/related;
  type="text/xml";
  start="F090DFD56421FD84AAF98C386AD50A44"; boundary="-----
  Part_0_27211574.1133404205718"
Accept: application/soap+xml, application/dime, multipart/related, text/*
User-Agent: Axis/1.1.2.19
Host: gmi-pc:1234
SOAPAction: "initiate"
Content-Length: 9307

------=_Part_0_27211574.1133404205718
Content-Type: text/xml;
  charset=UTF-8
Content-Transfer-Encoding: binary
Content-Id: <F090DFD56421FD84AAF98C386AD50A44>
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <initiate
      xmlns="">
      <payload xmlns="">
      <bin
        href="cid:1A744998BA8527BD121CAE96C022109F"
        xmlns=""></initiate
  </soapenv:Body>
</soapenv:Envelope>

------=_Part_0_27211574.1133404205718
Content-Type: application/octet-stream
Content-Transfer-Encoding: binary
Content-Id: <1A744998BA8527BD121CAE96C022109F>
```

See Also: The following documentation:

- [http://www.xml.com/pub/a/2003/02/26/binaryxml.html](http://www.xml.com/pub/a/2003/02/26/binaryxml.html) for XML, SOAP, and Binary Data
- [http://msdn.microsoft.com/mediamag/issues/02/12/DIME/default.aspx](http://msdn.microsoft.com/mediamag/issues/02/12/DIME/default.aspx) for MIME: Sending Files, Attachments, and SOAP Messages Via Direct Internet Message Encapsulation
- [http://gotdotnet.com/team/xml_wsspecs/dime/WSIDL-Extension-for-DIME.htm](http://gotdotnet.com/team/xml_wsspecs/dime/WSIDL-Extension-for-DIME.htm) for WSDL Extension for SOAP in DIME
- [http://www.w3.org/TR/wsdl.html](http://www.w3.org/TR/wsdl.html) for Web Services Description Language (WSDL) Specification 1.1

Displaying the Attachment Key for Binary Attachments Using the DIME Protocol in Oracle BPEL Control

The optSoapShortcut parameter value defaults to true in Oracle BPEL Control. This setting causes BPEL processes with SOAP message binary attachments that use the Direct Internet Message Encapsulation (DIME) protocol to not display their attachment
key in the Oracle BPEL Control audit trail for the process instance. This is because the binary attachment file is not saved to the dehydration database. Instead, an HTML file displays in the audit trail. For example:

```xml
<PutCompanyInfo>
  ...
  ...
  <report href="C:\orabpel\domains\default\tmp\bpe1 анаринретс\1.0.jar\report.html" />
</PutCompanyInfo>
```

As a workaround, set `optSoapShortcut` to `false` in Oracle BPEL Control. This enables the file to be saved to the dehydration store and the attachment key to display in the audit trail for the instance (instead of the HTML file). Copy and paste the attachment key into the Attachment Key field at the bottom of the audit trail window and click download to save it as a file for viewing. If you do this, note that the File Download message initially prompts you to save the attachment key as a JSP file type. Instead, save the file as an HTML file type.

Summary

This chapter provides an overview of the role of XML data in BPEL processes, including describing the large role that XPath expressions play in manipulating XML data.
Invoking a Synchronous Web Service

Synchronous Web services provide an immediate response to a query: BPEL can connect to synchronous Web services through a partner link, send data, and then receive the reply using a synchronous callback.

This chapter contains the following topics:
- Use Case for Synchronous Web Services
- Overview of Synchronous Service Concepts
- Calling a Synchronous Service
- Summary

Use Case for Synchronous Web Services

Using synchronous Web services is demonstrated in 104.SyncQuoteConsumer. This sample shows a BPEL process sending a stock code to a Web service and receiving a stock quote in return. It examines how synchronous functionality is defined in the stock quote Web service's CreditRatingService.wsdl file (the Web service to be called) and the client's QuoteConsumer.bpel file and bpel.xml deployment description file.

This chapter demonstrates how to establish a partner link and set up a synchronous callback. It discusses the components necessary to perform a synchronous callback, examines how these components are coded, and shows how to set up a synchronous callback.

See Also: The following files are used as examples in this chapter.
- SOA_Oracle_Home\bpel\samples\tutorials\104.SyncQuoteConsumer\bpel\QuoteConsumer.bpel
- SOA_Oracle_Home\bpel\samples\tutorials\104.SyncQuoteConsumer\bpel\bpel.xml
- SOA_Oracle_Home\bpel\samples\tutorials\104.SyncQuoteConsumer\bpel\QuoteConsumer.wsdl
- SOA_Oracle_Home\bpel\samples\utils\104.StockQuoteService\bpel\StockQuoteService.wsdl
Overview of Synchronous Service Concepts

A synchronous callback requires the following components:

- **Partner link**: Defines the location and the role of the Web services that the BPEL process connects to in order to perform tasks, as well as the variables used to carry information between the Web service and the BPEL process. A partner link is required for each Web service that the BPEL process calls.

- **Invoke activity**: Opens a port in the BPEL process to send and receive data. It uses this port to submit the required data and receive the response. In the credit rating service example, the invoke activity submits the stock code entered by the customer to the stock quote service and receives a stock quote in return. For synchronous callbacks, only one port is needed for both the send and receive functions.

Each domain has the attribute `syncMaxWaitTime`. This attribute has a default of 45 seconds, but can be reconfigured by the domain administrator. If the BPEL process does not receive a reply within the specified time, then the activity fails.

**See Also**: Oracle Application Server Performance Guide for additional details about `syncMaxWaitTime`

Establishing the Partner Link

This section covers the following topics:

- Defining the Partner Link in the BPEL Code
- Using the WSDL File to Enable the Web Services to Work with a BPEL Process
- Performing Lookups for Services that Use Partner Links
- Accessing Web Services on Remote Servers

**Defining the Partner Link in the BPEL Code**

In the BPEL code, the partner link defines the link name and type, and the role of the BPEL process in interacting with the partner service.

From the BPEL source code, the `StockQuoteService` partner link definition is as follows:

```xml
<partnerLinks>
  <!-- The 'client' role represents the requester of this service. It is used for callback. The location and correlation information associated with the client role are automatically set using WS-Addressing. -->
  <partnerLink name="client" partnerLinkType="samples:QuoteConsumer" myRole="QuoteConsumerProvider" partnerRole="QuoteConsumerRequester"/>
  <partnerLink name="StockQuoteService" partnerLinkType="services:StockQuoteService" partnerRole="StockQuoteServiceProvider"/>
</partnerLinks>
```

Following the partner link are global variable definitions that are accessible throughout the BPEL process. The types for these variables are defined in the WSDL for the process itself.

```xml
<variables>
  <!-- Reference to the message passed as input during initiation -->
  <variable name="input" messageType="tns:QuoteConsumerRequestMessage"/>
</variables>
```
Overview of Synchronous Service Concepts

Invoking a Synchronous Web Service

<variable name="output" messageType="tns:QuoteConsumerResultMessage"/>
<variable name="request" messageType="services:StockQuoteServiceRequest"/>
<variable name="response" messageType="services:StockQuoteServiceResponse"/>
</variables>

The WSDL file defines the interface to your BPEL process—the messages that it accepts and returns, operations that are supported, and other parameters.

Using the WSDL File to Enable the Web Services to Work with a BPEL Process

The Web service's QuoteConsumer.wsdl file contains two sections that enable it to work with BPEL processes:

- partnerLinkType Section of the QuoteConsumer.wsdl File
- portType Section of the QuoteConsumer.wsdl File

partnerLinkType Section of the QuoteConsumer.wsdl File

The partnerLinkType section of the QuoteConsumer.wsdl file defines the following characteristics of the conversion between a BPEL process and the loan application approver Web service:

- The role (operation) played by each
- The portType provided by each for receiving messages within the context of the conversation

<!-- PartnerLinkType definition -->
<!-- the QuoteConsumer partnerLinkType binds the service and requestor portType into an asynchronous conversation. -->
<plnk:partnerLinkType name="QuoteConsumer">
  <plnk:role name="QuoteConsumerProvider">
    <plnk:portType name="tns:QuoteConsumer"/>
  </plnk:role>
  <plnk:role name="QuoteConsumerRequester">
    <plnk:portType name="tns:QuoteConsumerCallback"/>
  </plnk:role>
</plnk:partnerLinkType>

portType Section of the QuoteConsumer.wsdl File

A port type is a collection of related operations implemented by a participant in a conversation. A port type defines what information is passed back and forth, the form of that information, and so forth. A synchronous callback requires only one port type that both sends a request and receives the response, while an asynchronous callback (one where the reply is not immediate) requires two port types, one to send the request, and another to receive the reply when it arrives.

View the portType section of the QuoteConsumer.wsdl file. This is the stock quote Web service to which the client submits the stock code that the customer has entered.

<!-- PortType definition -->

<-- portType implemented by the QuoteConsumer BPEL process -->

Invoking a Synchronous Web Service 4-3
Overview of Synchronous Service Concepts

Synchronous services have one port type. The port initiates the synchronous process and calls back the client with the response. In this example, the portType CreditRatingService receives the stock code and returns the stock quote.

Performing Lookups for Services that Use Partner Links

A Universal Description, Discovery, and Integration (UDDI) browser is provided for looking up services when creating a partner link. Web Services Inspection Language (WSIL) and UDDI assist in the publishing and discovery of services.

UDDI is a Web-based distributed directory that enables businesses to list themselves on the Internet and discover each other, similar to a traditional phone book’s yellow and white pages. The specification provides a high level of functionality through the sample object access protocol (SOAP) by specifically requiring an infrastructure to be deployed.

WSIL approaches service discovery in a decentralized fashion, where service description information can be distributed to any location using a simple extensible XML document format. Unlike UDDI, it is not concerned with business entity information, nor does it specify a particular service description format. WSIL works under the assumption that you are already familiar with the service provider, and relies on other service description mechanisms such as WSDL.

To access this registry when creating a partner link, you must first create a connection to the UDDI registry:

1. Right-click UDDI Registry in the Connection Navigator of Oracle JDeveloper.
2. Select New UDDI Registry Connection.
3. Follow the wizard steps to create a connection.

Accessing Web Services on Remote Servers

When creating a partner link, you can also select Web services on remote servers. To specify the remote location, edit

`$OA_Oracle_Home/bpel/system/services/install/config/inspection.wsil`

The Web service is then accessible from Oracle JDeveloper. Click the Service Explorer icon when creating a partner link on the Create Partner Link window. This displays the Service Explorer window, which enables you to select the remote Web service.

See Also: "PartnerLink" on page B-36
Calling a Synchronous Service

Using the Invoke Activity to Perform a Request

The invoke activity includes the request global input variable defined in the variables section. The credit rating Web service uses this request global input variable. This variable contains the customer’s social security number. The response variable contains the credit rating returned by the credit rating service.

```xml
<sequence>
  <!-- Receive input from requestor. Note: This maps to operation defined in QuoteConsumer.wsdl -->
  <receive name="receiveInput" partnerLink="client" portType="samples:QuoteConsumer" operation="initiate" variable="input" createInstance="yes"/>
  <assign>
    <copy>
      <from variable="input" part="payload" query="/tns:symbol"/>
      <to variable="request" part="symbol" query="/symbol"/>
    </copy>
  </assign>
  <!-- Generate content of output message based on the content of the input message. -->
  <invoke name="invokeStockQuoteService" partnerLink="StockQuoteService"/>
  <assign>
    <copy>
      <from variable="response" part="result" query="/result"/>
      <to variable="output" part="payload" query="/tns:result"/>
    </copy>
  </assign>
  <!-- Asynchronous callback to the requester. Note: the callback location and correlation id is transparently handled using WS-addressing. -->
  <invoke name="replyOutput" partnerLink="client" portType="samples:QuoteConsumerCallback" operation="onResult" inputVariable="output"/>
</sequence>
```

See Also: "Invoke Activity" on page B-14

Calling a Synchronous Service

This section examines a synchronous callback operation using the QuoteConsumer.bpel file. For a more step-by-step approach, see http://www.oracle.com/technology/bpel and download the files under Training Material.

Figure 4–1 shows the diagram of the QuoteConsumer.bpmn file, which defines a simple application with five activities.
Calling a Synchronous Service

The following actions take place:

1. The receiveInput receive activity receives input from the user (client), as defined in the QuoteConsuter.wsdl file.

2. The first assign activity packages the data from the client so that it can be accepted by the invokeStockQuote service.

3. The invokeStockQuoteService activity sends the repackaged data to the StockQuoteService service and receives a response.

4. A second assign activity repackages this response into a replyOutput activity so that it can be accepted by the client application.

5. The replyOutput activity sends the repackaged response back to the client.

The following BPEL code performs the synchronous callback:

```bpel
<assign>
  <copy>
      <from variable="input" part="payload" query="/tns:symbol"/>
      <to variable="request" part="symbol" query="/symbol"/>
  </copy>
</assign>

<invoke name="invokeStockQuoteService" partnerLink="StockQuoteService" portType="services:StockQuoteService" operation="process" inputVariable="request" outputVariable="response"/>
```
Summary

This chapter describes the concepts for a BPEL process that invokes a synchronous Web service and adds a partner link. This service takes a stock code as input from a client and synchronously returns a stock quote.
This chapter describes how to call an asynchronous Web service. Asynchronous messaging styles are very useful for environments in which a service, such as a loan processor, can take a long time to process a client request. Asynchronous services also provide a more reliable fault-tolerant and scalable architecture than synchronous services.

This chapter contains the following topics:

- Use Case for Asynchronous Web Services
- Overview of Asynchronous Callback Concepts
- Calling an Asynchronous Service
- Using Correlation Sets in an Asynchronous Service
- Summary

**Use Case for Asynchronous Web Services**

United Loan publishes an asynchronous Web service that processes a client’s loan application request and then returns a loan offer. This use case discusses how to integrate a BPEL process with this asynchronous loan application approver Web service.

This use case illustrates the key design concepts for requesting information from an asynchronous service, and then receiving the response. The asynchronous United Loan service in this example is another BPEL process. However, the same BPEL call can interact with any properly designed Web service. The target Web service WSDL file contains the information necessary to request and receive the desired information.

**Figure 5-1** provides an overview of how this BPEL process works with the asynchronous loan processor Web service.
For the asynchronous Web service, which is indicated within the dotted rectangle between the BPEL process’s receive and reply activities, the following actions take place:

1. An assign activity (prepare LoanApp) prepares the loan application.
2. An invoke activity (initiate service) initiates the loan request. The contents of this request are put into a request variable. This request variable is sent to the asynchronous loan processor Web service.

When the loan request is initiated, a correlation ID unique to the client and partner link initiating the request is also sent to the loan processor Web service. The correlation ID ensures that the correct loan offer response is returned to the corresponding loan application requester.

3. The loan processor Web service then sends the correct response to the receive activity (Wait for callback), which has been tracked by the correlation ID.
4. An assign activity (Read offer) reads the loan application offer.

The remaining sections in this chapter provide specific details about the asynchronous functionality shown in Figure 5–1.
Overview of Asynchronous Callback Concepts

This section examines how asynchronous functionality is defined in the loan application approver Web service's LoanService.wsdl file (the Web service to be called) and the client's LoanBroker.bpel file and bpel.xml deployment description file. It covers the following topics:

- partnerLinkTypes for Asynchronous Services
- Calling the Service from BPEL
- How the Invoke and Receive Activities Work
- Managing Multiple Active BPEL Process Instances Using Correlation Methods
- Using the Reply Activity to Send Messages in Response to a Receive Activity
- Using Dehydration Points to Maintain Long-Running Asynchronous Processes

See Also: The following files are used as examples in this chapter.

- SOA_Oracle_HOME\bpel\samples\utils\AsyncLoanService\LoanService.wsdl
- SOA_Oracle_HOME\bpel\samples\tutorials\105.AsyncCompositeLoanBroker\bpel\LoanBroker.bpel
- SOA_Oracle_HOME\bpel\samples\tutorials\105.AsyncCompositeLoanBroker\bpel\bpel.xml

partnerLinkTypes for Asynchronous Services

The following sections in the Web service's LoanService.wsdl file enable it to work with BPEL processes:

- portType Section of the LoanService.wsdl File
- partnerLinkType Section of the LoanService.wsdl File

See Also: "PartnerLink" on page B-36

portType Section of the LoanService.wsdl File

The portType section of the LoanService.wsdl file defines the ports to be used for the asynchronous service.

Asynchronous services have two port types. Each port type performs a one-way operation: one port type initiates the asynchronous process and the other calls back the client with the asynchronous response. In this example, the portType LoanService receives the client’s loan application request and the portType LoanServiceCallback asynchronously calls back the client with the loan offer response.
Overview of Asynchronous Callback Concepts

partnerLinkType Section of the LoanService.wsdl File
The partnerLinkType section of the LoanService.wsdl file defines the following characteristics of the conversation between the BPEL process and the loan application approver Web service:

- The role (operation) played by each
- The portType provided by each for receiving messages within the context of the conversation

Partner link types in asynchronous services have two roles: one for the Web service provider and one for the client requester.

In this conversation, the LoanServiceProvider role and LoanService portType are used for client request messages and the LoanServiceRequester role and LoanServiceCallback portType are used for asynchronously returning (calling back) response messages to the client.

Calling the Service from BPEL
To call the service from BPEL, you need the following files to define how the process interfaces with the Web service:

5-4 Oracle BPEL Process Manager Developer’s Guide
Overview of Asynchronous Callback Concepts

Invoking an Asynchronous Web Service

- Partner Links Section in the .bpel File
- Deployment Descriptor File

Partner Links Section in the .bpel File

View the partnerLinks section of the LoanBroker.bpel file. The services with which a process interacts are designed as partner links. Each partner link is characterized by a partnerLinkType.

Each partner link is named. This name is used for all service interactions through that partner link. This is critical in correlating responses to different partner links for simultaneous requests of the same type.

Asynchronous processes use a second partner link for the callback to the client. In this example, the second partner link, LoanService, is used by the loan application approver Web service.

<!-- This process invokes the asynchronous LoanService. -->

```xml
<partnerLink name="LoanService" partnerLinkType="services:LoanService" myRole="LoanServiceRequester" partnerRole="LoanServiceProvider"/>
</partnerLinks>
```

The attribute myRole indicates the role of the client. The attribute partnerRole indicates the role of the partner in this conversation. Each partnerLinkType has a myRole and partnerRole attribute in asynchronous processes.

Deployment Descriptor File

Open the bpel.xml deployment descriptor file of samples\tutorials\105.AsyncCompositeLoanBroker. The loan application approver Web service appears. This properties id information is added to the file when you create a second partner link type.

```xml
<?xml version="1.0"?>
<bpel-process id="LoanBroker" src="LoanBroker.bpel" wsdlLocation="LoanBroker.wsdl">
<properties id="LoanService">
  <property name="wsdlLocation">
  </property>
</properties>
</bpel-process>
```

See Also:
- "Step 1: Adding a Partner Link for an Asynchronous Service" on page 5-11 for instructions on creating a partner link
- Appendix C, "Deployment Descriptor Properties"

How the Invoke and Receive Activities Work

View the variables and sequence sections of the LoanBroker.bpel file. Two areas of particular interest concern the invoke and receive activities:

- An invoke activity invokes a synchronous Web service (as discussed in Chapter 4, "Invoking a Synchronous Web Service") or initiates an asynchronous service.
  The invoke activity includes the request global input variable defined in the variables section. The request global input variable is used by the loan
application approver Web service. This variable contains the contents of the initial loan application request document.

- A receive activity that waits for the asynchronous callback from the loan application approver Web service. The receive activity includes the response global output variable defined in the variables section. This variable contains the loan offer response. The receive activity asynchronously waits for a callback message from a service. While the BPEL process is waiting, it is dehydrated, or compressed and stored, until the callback message arrives.

```xml
<variables>
  <variable name="request" messageType="services:LoanServiceRequestMessage"/>
  <variable name="response" messageType="services:LoanServiceResultMessage"/>
</variables>
```

<sequence>

<!-- initialize the input of LoanService -->
<assign>
 <!-- initiate the remote process -->
 <invoke name="invoke" partnerLink="LoanService"
 portType="services:LoanService"
 operation="initiate" inputVariable="request"/>

<!-- receive the result of the remote process -->
<receive name="receive_invoke" partnerLink="LoanService"
 portType="services:LoanServiceCallback"
 operation="onResult" variable="response"/>

When an asynchronous service is initiated with the invoke activity, a correlation ID unique to the client request is also sent, using WS-Addressing (described in "WS-Addressing" on page 5-7). Because multiple processes may be waiting for service callbacks, Oracle BPEL Server must know which BPEL process instance is waiting for a callback message from the loan application approver Web service. The correlation ID enables Oracle BPEL Server to correlate the response with the appropriate requesting instance.

**See Also:** The following sections for instructions on creating invoke and receive activities:

- "Step 2: Adding an Invoke Activity" on page 5-12
- "Step 3: Adding a Receive Activity" on page 5-13
- "Invoke Activity" on page B-14
- "Receive Activity" on page B-20

**Using the createInstance Attribute to Start a New Instance**

You may have noticed a createInstance attribute in the initial receive activity of the sequence section of the LoanBroker.bpel file. In this initial receive activity, the createInstance element is set to yes. This starts a new instance of the BPEL process. At least one instance startup is required for a conversation. For this reason, you set the createInstance variable to no in the second receive activity.

The source code for the createInstance attribute appears as follows:
Overview of Asynchronous Callback Concepts

Managing Multiple Active BPEL Process Instances Using Correlation Methods

Because there can be many active instances at any given point in time, Oracle BPEL Server must be able to direct Web service responses to the correct BPEL process instance. You can use the following correlation methods to identify asynchronous messages to ensure that asynchronous callbacks locate the appropriate client:

- **WS-Addressing**
- **Using Correlation Sets to Coordinate Asynchronous Message Body Contents**

**WS-Addressing**

Web Services Addressing (WS-Addressing) is a public specification and is the default correlation method supported by Oracle BPEL Process Manager. You do not need to edit the .bpel and .wsdl files to use WS-Addressing. WS-Addressing uses simple object access protocol (SOAP) headers for asynchronous message correlation. Messages are independent of the transport or application used. Figure 5-2 provides an overview.

*Figure 5-2 Callback with WS-Addressing Headers*

Figure 5-2 shows how messages are passed along with WS headers so that the response can be sent to the correct destination.
Overview of Asynchronous Callback Concepts

The example in this chapter uses WS-Addressing for correlation. To view the messages, you can use TCP tunneling, which is described in "Using TCP Tunneling to See Messages Exchanged Between Programs" on page 5-8.

WS-Addressing defines the following information typically provided by transport protocols and messaging systems. This information is processed independently of the transport or application:

- **Endpoint location (reply-to address):** The reply-to address specifies the location at which a BPEL client is listening for a callback message.

- **Conversation ID:** Use TCP tunneling to view SOAP messages exchanged between the BPEL process flow and the Web service (including those containing the correlation ID). You can see the exact SOAP messages that are sent to, or received from, services with which a BPEL process flow communicates.

You insert a software listener between your BPEL process flow and the Web service. Your BPEL process flow communicates with the listener (called a TCP tunnel). The listener forwards your messages to the Web service, and also displays them. Responses from the Web service are returned to the tunnel, which displays and forwards them back to the BPEL process.

**Using TCP Tunneling to See Messages Exchanged Between Programs**

The messages that are exchanged between programs and services can be seen through TCP tunneling. This is particularly useful with Web services and BPEL processes when you want to see the exact SOAP messages exchanged between the BPEL process flow and Web services.

To monitor the SOAP messages, insert a software listener between your flow and the service. Your flow communicates with the listener (called a TCP tunnel) and the listener forwards your messages to the service, as well as displaying them. Likewise, responses from the service are returned to the tunnel, which displays them and then forwards them back to the flow.

To see all the messages exchanged between Oracle BPEL Server and a Web service, you need only a single TCP tunnel for synchronous services because all the pertinent messages are communicated in a single request and reply interaction with the service. For asynchronous services, you must set up two tunnels, one for the invocation of the service and another for the callback port of the flow.

**Setting up a TCP Listener for Synchronous Services**

Follow these steps to set up a TCP listener for synchronous services initiated by an Oracle BPEL Process Manager process:

1. Start your TCP listener to listen on a port such as 1234 and send on a port such as 9700 (port 9700 is used in this example and is the default after Oracle BPEL Process Manager for Developers installation). If you installed Oracle BPEL Process Manager as part of an Oracle Application Server SOA install type, substitute the correct port number throughout these instructions. For example, you can use the TCP tunnel included with Apache Axis (bundled with Oracle BPEL Process Manager) by executing the following from the operating system command prompt:

   prompt> obsetenv
   prompt> java -classpath %OB_CLASSPATH% orabpel.apache.axis.utils.tcpmon 1234 localhost 9700

2. Add a location property in the bpel.xml deployment descriptor file for your flow to override the endpoint of the service. For example, to see the messages
Overview of Asynchronous Callback Concepts

Exchanged between the LoanFlow demo sample and the CreditRatingService that it calls, change the definition of the CreditRatingService location as shown below in the LoanFlow deployment descriptor in SOA_Oracle_Home/bpel/samples/demos/LoanDemo/LoanFlow/bpel.xml:

```xml
<partnerLinkBinding name="creditRatingService">
  <property name="wsdlLocation">
    http://localhost:9700/orabpel/default/CreditRatingService/CreditRatingService?wsdl
  </property>
  <property name="location">
    http://localhost:1234/orabpel/default/CreditRatingService
  </property>
</partnerLinkBinding>
```

3. Compile and deploy the LoanDemo from the operating system command prompt:

```bash
prompt> cd SOA_Oracle_Home/bpel/samples/demos/LoanDemo
prompt> ant
```

Note that while the CreditRatingService is also a BPEL process, the same technique can be used to see the SOAP messages passed to invoke a BPEL process as a Web service from another tool kit such as Axis or .NET.

See Also: The TCP Monitor tool located in the following directory:

SOA_Oracle_Home/bpel/bin/obtunnel.bat

Setting up a TCP Listener for Asynchronous Services

Follow these steps to set up a TCP listener to display the SOAP messages for callbacks from asynchronous services:

1. Start a TCP listener to listen on a port such as 9710 and to send on the Oracle BPEL Process Manager port (for example, 9700 is the default after installation of Oracle BPEL Process Manager for Developers).

2. Turn off the optimization of local SOAP calls performed by Oracle BPEL Process Manager to see the impact of changing the callback port:
   a. Click Manage BPEL Domain in the upper right of Oracle BPEL Control.
   b. Scroll down to the optSoapShortcut property.
   c. Change the value from true to false.

3. Access Oracle BPEL Admin Console at:

```bash
http://localhost:port/BPELAdmin
```

4. Scroll down to the SoapServerUrl property on the Configuration tab.


6. Click the Apply button.

7. Restart Oracle BPEL Server to initialize these changes and initiate any flow that invokes asynchronous Web services (for example the LoanFlow demonstration). You can combine this with the synchronous TCP tunneling configuration to send the UnitedLoan service initiation request through your first TCP tunnel. The callbacks from the asynchronous services are shown in the TCP listener, such as the UnitedLoan service callback.

If you are an Oracle JDeveloper user, you can also use the built-in Packet Monitor to see SOAP messages for both synchronous and asynchronous services.
Overview of Asynchronous Callback Concepts

Using Correlation Sets to Coordinate Asynchronous Message Body Contents
Correlation sets are a BPEL mechanism that provides for the correlation of asynchronous messages based on message body contents. To use this method, define the correlation sets in your .bpel file. This method is designed for services that do not support WS-Addressing or for certain sophisticated conversation patterns, for example, when the conversation is in the form A > B > C > A instead of A > B > A.

See Also: The following correlation set examples:
- "Using Correlation Sets in an Asynchronous Service" on page 5-14 for a tutorial on creating correlations sets in Oracle JDeveloper
- SOA_Oracle_Home\bpel\samples\tutorials\109.CorrelationSets

Using the Reply Activity to Send Messages in Response to a Receive Activity
The reply activity enables the business process to send a message in reply to a message that was received through a receive activity. The combination of a receive and a reply forms a request-response operation on the WSDL portType for the process.

```xml
<reply partnerLink="ncname" portType="qname" operation="ncname" variable="ncname"? faultName="qname"? standard-attributes>
  <correlations>?
    <correlation set="ncname" initiate="yes|no">+</n</correlations>
</reply>
```

See Also:
- "Returning External Faults" on page 8-7
- "Reply Activity" on page B-21
- SOA_Oracle_Home\bpel\samples\references\Reply

Using Dehydration Points to Maintain Long-Running Asynchronous Processes
To automatically maintain long-running asynchronous processes and their current state information in a database while they wait for asynchronous callbacks, you use a database as a dehydration store. Storing the process in a database preserves any loss of state or reliability if a system shuts down or a network problem occurs. This feature increases both BPEL process reliability and scalability. You can also use it to support clustering and failover.

You insert this point between the invoke activity and receive activity. Figure 5–1 on page 5-2 shows an example of a dehydration point in the loan application approver Web service.

See Also:
- The following correlation set examples:
Calling an Asynchronous Service

To add asynchronous functionality to a BPEL process, complete the tasks in this section:

- Step 1: Adding a Partner Link for an Asynchronous Service
- Step 2: Adding an Invoke Activity
- Step 3: Adding a Receive Activity
- Step 4: Performing Additional Activities

Step 1: Adding a Partner Link for an Asynchronous Service

These instructions describe how to create a partner link named LoanService for the loan application approver Web service.

1. Double-click LoanBroker.bpel in the Application Navigator.
2. In the diagram window, right-click either side of the BPEL process (under Services).
3. Select Create Partner Link.
   The Create Partner Link window appears.
4. Enter the following details to create a second partner type and select the loan application approver Web service:
   - Name: Enter a name for the partner link.
   - Process: The BPEL process name
   - WSDL File: Enter the name of the WSDL file to use. Click the Service Explorer icon above this field to locate the correct WSDL.
   - Partner Link Type: Refers to the external service with which the BPEL process is to interface. Select from the list.
   - Partner Role: Refers to the role of the external source, for example, provider. Select from the list.
   - My Role: Refers to role of the BPEL process in this interaction, for example, requester. Select from the list.
5. Click OK.
Calling an Asynchronous Service

A new partner link for the loan application approver Web service (United Loan) appears in the Services area of the .bpel file’s diagram window.

See Also:
- “partnerLinkTypes for Asynchronous Services” on page 5-3 for conceptual details about partner links
- “PartnerLink” on page B-36

Step 2: Adding an Invoke Activity

Follow these instructions to create an invoke activity and a global input variable named request. This activity initiates asynchronous BPEL process activity with the loan application approver Web service (United Loan). The loan application approver Web service uses the request input variable to receive the loan request from the client.

1. Drag an invoke activity from the Component Palette to beneath the receive activity.
2. In the .bpel file’s diagram window, right-click either side of the BPEL process and select View > Variables from the menu. The Variables window appears.
3. In the Variables window, select the second Variables folder in the navigation tree, and click Create. The Create Variable dialog box appears.
4. Enter the variable name and select Message Type from the options provided:
   - Simple Type: This option lets you select an XML schema simple type, for example, string, Boolean, and so on.
   - Message Type: This option enables you to select a WSDL message file definition of a partner link or of the project WSDL file of the current BPEL process (for example, a response message or a request message). You can specify variables associated with message types as input or output variables for invoke, receive, or reply activities.
     To display the message type, select the Message Type option, and then select its flashlight icon to display the Type Chooser window. From here, expand the Message Types navigation window to select Message Types > Partner Links > Loan Service > United Loan > Message Types > LoanServiceRequestMessage.
   - Element: This option lets you select an XML schema element of the project schema file or project WSDL file of the current BPEL process, or of a partner link.

5-12  Oracle BPEL Process Manager Developer’s Guide
5. Click OK, then click Close.

6. Double-click the invoke activity to display the Invoke window.

7. In the Invoke window, select the LoanService partner link from the Partner Link list and initiate from the Operation list.

8. Select the input variable you created in Step 4, by clicking the second icon to the right of the Input Variable field.

   The Variable Chooser window appears, where you can select the variable.

   There is no output variable specified because the output variable is returned in the receive operation. The invoke activity and the global input variable are created.

   **See Also:**
   - “How the Invoke and Receive Activities Work” on page 5-5 for conceptual details about the invoke activity
   - “Invoke Activity” on page B-14

9. Click OK

**Step 3: Adding a Receive Activity**

Follow these steps to create a receive activity and a global output variable named response. This activity waits for the loan application approver Web service’s callback operation. The loan application approver Web service uses this output variable to send the loan offer result to the client.

1. From the Component Palette, drag a receive activity to the location right after the invoke activity you created in “Step 2: Adding an Invoke Activity” on page 5-12.

2. Create a variable to hold the receive information by invoking the Create Variable window, as you did in Step 2 through Step 5, starting on page 5-12.
Using Correlation Sets in an Asynchronous Service

3. Double-click the receive activity and change its name to receive_invoke.
4. Select LoanService from the Partner Link list and onResult from the Operation list. Do not select the Create Instance check box.
5. Select the variable you created in Step 2 through Step 5, starting on page 5-12.
6. Click OK.

The receive activity and the output variable are created. Because the initial receive activity in the LoanBroker.bpel file created the initial BPEL process instance, a second instance does not need to be created.

See Also: "Receive Activity" on page B-20

Step 4: Performing Additional Activities

In addition to the asynchronous-specific tasks, you must perform the following tasks.

- Create an initial assign activity for data manipulation in front of the invoke activity that copies the client’s input variable loan application request document payload into the loan application approver Web service’s request variable payload.
- Create a second assign activity for data manipulation after the receive activity that copies the loan application approver Web service’s response variable loan application results payload into the output variable for the client to receive.

See Also: The following documentation for information on creating and defining an assign activity:
- Oracle BPEL Process Manager Quick Start Guide
- Oracle BPEL Process Manager Order Booking Tutorial

Using Correlation Sets in an Asynchronous Service

This tutorial describes how to use correlation sets in an asynchronous service with Oracle JDeveloper. Correlation sets enable you to correlate asynchronous messages based on message body contents. You define correlation sets when interactions are not simple invoke-receive activities. This example illustrates how to use correlation sets for a process having three receive activities with no associated invoke activities. This section contains the following topics:
Using Correlation Sets in an Asynchronous Service

- Step 1: Creating a Project
- Step 2: Configuring Partner Links and File Adapter Services
- Step 3: Creating Three Receive Activities
- Step 4: Creating Correlation Sets
- Step 5: Associating Correlation Sets with Receive Activities
- Step 6: Creating Property Aliases
- Step 7: Reviewing WSDL File Content

Step 1: Creating a Project

1. Right-click your application in the Application Navigator section of the designer window.
2. Select New Project.
3. Double-click BPEL Process Project in the Items window to display the BPEL Project Creation Wizard window.
4. Enter an appropriate name in the Name field (for this example, MyCorrelationSet is used).
5. Select Asynchronous BPEL Process from the Template list.
6. Click Finish.

Step 2: Configuring Partner Links and File Adapter Services

You now create three partner links that use the adapter services.

This section contains these topics:
- Creating an Initial Partner Link and File Adapter Service
  You create an initial partner link with an adapter service for reading a loan application.
- Creating a Second Partner Link and File Adapter Service
  You create a second partner link with an adapter service for reading an application response.
- Creating a Third Partner Link and File Adapter Service
  You create a third partner link with an adapter service for reading a customer response.

Creating an Initial Partner Link and File Adapter Service

1. Select Services from the Component Palette.
2. Drag and drop an initial PartnerLink activity onto the right side of the designer window anywhere beneath the header Services.
3. Enter FirstReceivePL in the Name field.
4. Click the third icon at the top (the Define Adapter Service icon). This starts the Adapter Configuration Wizard.
Using Correlation Sets in an Asynchronous Service

5. Click Next on the Welcome window.
6. Select File Adapter on the Adapter Type window and click Next.
7. Enter FirstReceive in the Service Name field on the Service Name window and click Next.
8. Select Read File as the Operation Type on the Operation window and click Next. The Operation Name field is automatically filled in with Read.
9. Select Directory Names are Specified as Physical Path.
10. Click Browse next to the Directory for Incoming Files (physical path) field.
11. Select a directory from which to read files (for this example, C:\files\receiveprocess\FirstInputDir is selected).
12. Click Select.
13. Click Next.
14. Enter appropriate file filtering parameters in the File Filtering window.
15. Click Next.
16. Enter appropriate file polling parameters in the File Polling window.
17. Click Next.
18. Click Browse next to the Schema Location field in the Messages window to display the Type Chooser window.
19. Select an appropriate XSD schema file. For this example, Book1_4.xsd is the schema and LoanAppl is the schema element selected.
20. Click OK.
   The Schema Location field (Book1_4.xsd for this example) and the Schema Element field (LoanAppl for this example) are filled in.
21. Click Next.
22. Click Finish.

You are returned to the Partner Link window. All other fields are automatically completed. The window looks as follows:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>FirstReceive</td>
</tr>
<tr>
<td>WSDL File</td>
<td>file://c:\OraJDev\jdev\mywork\myapplication\MyCorrelationSet\bpel\F</td>
</tr>
<tr>
<td></td>
<td>initReceive.wsdl</td>
</tr>
<tr>
<td></td>
<td>where c:\OraJDev represents the Oracle JDeveloper home directory for</td>
</tr>
<tr>
<td></td>
<td>this example.</td>
</tr>
<tr>
<td>Partner Link Type</td>
<td>Read_plt</td>
</tr>
<tr>
<td>Partner Role</td>
<td>Leave unspecified.</td>
</tr>
<tr>
<td>My Role</td>
<td>Read_role</td>
</tr>
</tbody>
</table>
Creating a Second Partner Link and File Adapter Service

1. Drag and drop a second PartnerLink activity below the FirstReceivePL partner link activity.
2. Enter SecondReceivePL in the Name field.
3. Click the third icon at the top (the Define Adapter Service icon).
4. Click Next on the Welcome window.
5. Select File Adapter in the Adapter Type window and click Next.
6. Enter SecondFileRead in the Service Name field on the Service Name window and click Next. This name must be unique from the one you entered in Step 7 on page 5-16.
7. Select Read File as the Operation Type in the Operation window.
8. Change the name in the Operation Name field to Read1.
9. Click Next.
10. Select Directory Names are Specified as Physical Path.
11. Click Browse next to the Directory for Incoming Files (physical path) field.
12. Select a directory from which to read files (for this example, C:\files\receiveprocess\SecondInputDir is entered).
13. Click Select.
14. Click Next.
15. Enter appropriate file filtering parameters in the File Filtering window.
16. Click Next.
17. Enter appropriate file polling parameters in the File Polling window.
18. Click Next.
19. Click Browse next to the Schema Location field in the Messages window to display the Type Chooser window.
20. Select an appropriate XSD schema file. For this example, Book1_5.xsd is the schema and LoanAppResponse is the schema element selected.
21. Click OK.
   The Schema Location field (Book1_5.xsd for this example) and the Schema Element field (LoanAppResponse for this example) are filled in.
22. Click Next.
23. Click Finish.

You are returned to the Partner Link window. All other fields are automatically completed. The window looks as follows:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>SecondReceive</td>
</tr>
</tbody>
</table>
Using Correlation Sets in an Asynchronous Service

Creating a Third Partner Link and File Adapter Service

1. Drag and drop a third PartnerLink activity below the SecondReceivePL partner link activity.
2. Enter ThirdReceivePL in the Name field.
3. Click the third icon at the top (the Define Adapter Service icon).
4. Click Next on the Welcome window.
5. Select File Adapter in the Adapter Type window and click Next.
6. Enter ThirdFileRead in the Service Name field of the Service Name window and click Next. This name must be unique from the one you entered in Step 7 on page 5-16 and Step 6 on page 5-17.
7. Select Read File as the Operation Type in the Operation window
8. Change the name in the Operation Name field to Read2. This name must be unique.
9. Click Next.
10. Select Directory Names are Specified as Physical Path.
11. Click Browse next to the Directory for Incoming Files (physical path) field.
12. Select a directory from which to read files (for this example, C:\files\receiveprocess\ThirdInputDir is entered).
13. Click Select.
14. Click Next.
15. Enter appropriate file filtering parameters in the File Filtering window.
16. Click Next.
17. Enter appropriate file polling parameters in the File Polling window.
18. Click Next.
19. Click Browse next to the Schema Location field in the Messages window to display the Type Chooser window.
20. Select an appropriate XSD schema file. For this example, Book1_6.xsd is the schema and CustResponse is the schema element selected.
21. Click OK.
   The Schema Location field (Book1_6.xsd for this example) and the Schema Element field (CustResponse for this example) are filled in.

Field | Value
--- | ---
WSDL File | file://OraJDev/jdev/mynwork/mypapplication/MyCorrelationSet/bpel/SecondFileRead.wsdl where c:/OraJDev represents the Oracle JDeveloper home directory for this example.
Partner Link Type | Read1.plt
Partner Role | Leave unspecified.
My Role | Read1_role

24. Click OK.
22. Click Next.
23. Click Finish.

You are returned to the Partner Link window. All other fields are automatically completed. The window looks as follows:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>ThirdReceive</td>
</tr>
<tr>
<td>WSDL File</td>
<td>file://c/OraJDev/jdev/mywork/myapplication/MyCorrelationSet/bpel/ThirdFileRead.wsdl where c/OraJDev represents the Oracle JDeveloper home directory for this example.</td>
</tr>
<tr>
<td>Partner Link Type</td>
<td>Read2_plt</td>
</tr>
<tr>
<td>Partner Role</td>
<td>Leave unspecified.</td>
</tr>
<tr>
<td>My Role</td>
<td>Read2_role</td>
</tr>
</tbody>
</table>

24. Click OK.

When complete, the designer window looks as follows:

Step 3: Creating Three Receive Activities
You now create three receive activities; one for each partner link. The receive activities specify the partner link from which to receive information.

This section contains the following topics:
- Creating an Initial Receive Activity
- Creating a Second Receive Activity
- Creating a Third Receive Activity

Creating an Initial Receive Activity
1. Drag and drop a Receive activity from the Process Activities list of the Component Palette section into the designer window.
2. Double-click the receive icon to display the Receive window.
3. Enter the following details to associate the first partner link (FirstReceivePL) with the first receive activity:
Using Correlation Sets in an Asynchronous Service

4. Click the first icon to the right of the **Variable** field. This is the automatic variable creation icon.
5. Click **OK** on the Create Variable window that appears.
   A variable named `receiveFirst_Read1_InputVariable` is automatically created in the **Variable** field.
6. Ensure that you selected the **Create Instance** check box, as mentioned in Step 3.
7. Click **OK**.

**Creating a Second Receive Activity**

1. Drag and drop a second **Receive** activity from the **Component Palette** section to below the **receiveFirst receive** activity.
2. Double-click the **receive** icon to display the Receive window.
3. Enter the following details to associate the second partner link (**SecondReceivePL**) with the second receive activity:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>receiveSecond</td>
</tr>
<tr>
<td>Partner Link</td>
<td>SecondReceivePL</td>
</tr>
<tr>
<td>Create Instance</td>
<td>Do not select this check box.</td>
</tr>
</tbody>
</table>

   The **Operation (Read1)** field is automatically filled in.
4. Click the first icon to the right of the **Variable** field.
5. Click **OK** on the Create Variable window that appears.
   A variable named `receiveSecond_Read1_InputVariable` is automatically created in the **Variable** field.
6. Click **OK**.

**Creating a Third Receive Activity**

1. Drag and drop a third **Receive** activity from the **Component Palette** section to below the **receiveSecond receive** activity.
2. Double-click the **receive** icon to display the Receive window.
3. Enter the following details to associate the third partner link (**ThirdReceivePL**) with the third receive activity:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>receiveThird</td>
</tr>
</tbody>
</table>

5-20  Oracle BPEL Process Manager Developer's Guide
Using Correlation Sets in an Asynchronous Service

Invoking an Asynchronous Web Service

---

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Partner Link</td>
<td>ThirdReceivePL</td>
</tr>
<tr>
<td>Create Instance</td>
<td>Do not select this check box.</td>
</tr>
</tbody>
</table>

The Operation (Read2) field is automatically filled in.

4. Click the first icon to the right of the Variable field.
5. Click OK on the Create Variable window that appears.
   A variable named receiveThird_Read2_InputVariable is automatically created in the Variable field.
6. Click OK.
   Each receive activity is now associated with a specific partner link.

**Step 4: Creating Correlation Sets**

You now create correlation sets. A set of correlation tokens is a set of properties shared by all messages in the correlated group.

This section contains the following topics:

- Creating an Initial Correlation Set
- Creating a Second Correlation Set

**Creating an Initial Correlation Set**

1. Right-click Correlation Sets and select Expand All Child Nodes in the Structure window of Oracle JDeveloper.
2. Right-click Correlation Sets and select Create Correlation Set.
3. Enter CorrelationSet1 in the Name field of the Create Correlation Set window.
4. Click Add in the Properties section to display the Property Chooser window.
5. Select Properties, then click Create (first icon at the top) to display the Create Correlation Set Property window.
6. Enter NameCorr in the Name field and click the flashlight icon to the right of the Type field.
7. Select string in the Type Chooser window and click OK.
8. Click OK to close the Create Correlation Set Property window, the Property Chooser window, and the Create Correlation Set window.

**Creating a Second Correlation Set**

1. Return to the Correlation Sets section in the Structure window of Oracle JDeveloper.
2. Right-click Correlation Sets and select Create Correlation Set.
3. Enter CorrelationSet2 in the Name field of the Create Correlation Set window.
4. Click Add in the Properties section to display the Property Chooser window.
5. Select Properties, then click Create to display the Create Correlation Set Property window.
Using Correlation Sets in an Asynchronous Service

6. Enter IDCorr in the Name field and click the flashlight icon to the right of the Type field.
7. Select double in the Type Chooser window and click OK.
8. Click OK to close the Create Correlation Set Property window, the Property Chooser window, and the Create Correlation Set window.

Step 5: Associating Correlation Sets with Receive Activities

You now associate the correlation sets with the receive activities. You perform the following correlation set tasks:

- For the first correlated group, the first and second receive activities are correlated with the CorrelationSet1 correlation set.
- For the second correlated group, the second and third receive activities are correlated with the CorrelationSet2 correlation set.

This section contains the following topics:
- Associating the First Correlation Set with a Receive Activity
- Associating the Second Correlation Set with a Receive Activity
- Associating the Third Correlation Set with a Receive Activity

Associating the First Correlation Set with a Receive Activity

1. Double-click the receiveFirst receive activity to display the Receive window.
2. Click the Correlations tab.
3. Click Add, select CorrelationSet1, then click OK.
4. Set the Initiate column to yes. When set to yes, the set is initiated with the values of the properties occurring in the message being exchanged.
5. Click OK.

Associating the Second Correlation Set with a Receive Activity

1. Double-click the receiveSecond receive activity to display the Receive window.
2. Click the Correlations tab.
3. Click Add, select CorrelationSet2, then click OK.
4. Set the Initiate column to yes.
5. Click Add and select CorrelationSet1.
6. Click OK.
7. Set the Initiate column to no for CorrelationSet1.

8. Click OK.
   This groups the first and second receive activities into a correlated group.

Associating the Third Correlation Set with a Receive Activity
1. Double-click the receiveThird receive activity to display the Receive window.
2. Click the Correlations tab.
3. Click Add and select CorrelationSet2.
4. Click OK.
5. Set the Initiate column to no for CorrelationSet2.

6. Click OK.
   This groups the second and third receive activities into a second correlated group.
Using Correlation Sets in an Asynchronous Service

Step 6: Creating Property Aliases

Property aliases enable you to map a global property to a field in a specific message part. This enables the property name to become an alias for the message part and location. The alias can be used in XPath expressions.

This section contains the following topics:

■ Creating Property Aliases for NameCorr
■ Creating Property Aliases for IDCorr

Creating Property Aliases for NameCorr

You create the following two property aliases for the NameCorr correlation set.

■ Map NameCorr to the LoanAppl message type part of the receiveFirst receive activity. This receive activity is associated with the FirstReceivePL partner link (defined by the FirstReceive.wsdl file).

■ Map NameCorr to the incoming LoanAppResponse message type part of the receiveSecond receive activity. This receive activity is associated with the SecondReceivePL partner link (defined by the SecondFileRead.wsdl file).

1. Right-click Property Aliases in the Structure section of Oracle JDeveloper.
2. Select Create Property Alias.
3. Select NameCorr in the Property list.
4. Expand and select Message Types > Partner Links > FirstReceivePL > FirstReceive.wsdl > Message Types > LoanAppl_msg > Part - LoanAppl
5. Press Ctrl and then the space bar in the Query field to define the following XPath expression:
   
   /ns2:LoanAppl/ns2:Name

6. Click OK.
7. Repeat Step 1 through Step 3 to create a second property alias for NameCorr.
9. Press Ctrl and then the space bar in the Query field to define the following XPath expression:
   
   /ns4:LoanAppResponse/ns4:APR

Creating Property Aliases for IDCorr

You create the following two property aliases for the IDCorr correlation set.

■ Map IDCorr to the LoanAppResponse message type part of the receiveSecond receive activity. This receive activity is associated with the SecondReceivePL partner link (defined by the SecondFileRead.wsdl file).

■ Map IDCorr to the CustResponse message type part of the receiveThird receive activity. This receive activity is associated with the ThirdReceivePL partner link (defined by the ThirdFileRead.wsdl file).

1. Right-click Property Aliases in the Structure section.
2. Select Create Property Alias.
3. Select IDCorr in the Property list.
5. Press Ctrl and then the space bar in the Query field to define the following XPath expression:
   /ns4:LoanAppResponse/ns4:APR
6. Click OK.
7. Repeat Step 1 through Step 3 to create a second property alias for IDCorr.
9. Press Ctrl and then the space bar in the Query field to define the following XPath expression:
   /ns6:CustResponse/ns6:APR

Design is now complete.

Step 7: Reviewing WSDL File Content

The NameCorr and IDCorr correlation set properties are defined in the MyCorrelationSet_Properties.wsdl file in the Application Navigator of Oracle JDeveloper:

```xml
<definitions
  name="properties"
  targetNamespace="http://xmlns.oracle.com/MyCorrelationSet/correlationset"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partner-link/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <bpws:property name="NameCorr" type="xsd:string"/>
  <bpws:property name="IDCorr" type="xsd:double"/>
</definitions>
```

The property aliases are defined in the MyCorrelationSet.wsdl file:

```xml
<bpws:propertyAlias propertyName="ns1:NameCorr" messageType="ns3:LoanAppl_msg" part="LoanAppl" query="/ns2:LoanAppl/ns2:Name"/>
<bpws:propertyAlias propertyName="ns1:NameCorr" messageType="ns5:LoanAppResponse_msg" part="LoanAppResponse" query="/ns4:LoanAppResponse/ns4:APR"/>
<bpws:propertyAlias propertyName="ns1:IDCorr" messageType="ns5:LoanAppResponse_msg" part="LoanAppResponse" query="/ns4:LoanAppResponse/ns4:APR"/>
<bpws:propertyAlias propertyName="ns1:IDCorr" messageType="ns7:CustResponse_msg" part="CustResponse" query="/ns6:CustResponse/ns6:APR"/>
```

Because the BPEL process is not created as a Web services provider in this example, the MyCorrelationSet.wsdl file is not referenced in the BPEL process. Therefore, you must import the MyCorrelationSet.wsdl file inside the FirstReceive.wsdl file to reference the correlation sets:
<import namespace="http://xmlns.oracle.com/MyCorrelationSet"
location="MyCorrelationSet.wsdl"/>

Summary

This chapter describes the concepts for a BPEL process that invokes an asynchronous Web service. This service takes a loan application request document as input from a client and asynchronously returns an approved loan offer. An example of how to create correlation sets in Oracle JDeveloper is also provided.
Parallel flows enable a BPEL process to perform multiple tasks at the same time, which is especially useful when you need to perform several time-consuming and independent tasks.

This chapter contains the following topics:
- Use Case for Parallel Flows
- Overview of Parallel Flow Concepts
- Defining a Parallel Flow
- Customizing the Number of Flow Activities by Using the flowN Activity
- Summary

**Use Case for Parallel Flows**

In Chapter 5, "Invoking an Asynchronous Web Service" you learned how to call an asynchronous Web service for the United Loan service. Because the United Loan service can take up to several days to return a loan offer but you need to collect another loan offer from Star Loan, you can define your BPEL process so both tasks run in parallel.

This use case shows how to program the BPEL flow to perform two asynchronous callbacks to loan services in parallel.

**See Also:** The following sample:
- SOA_Oracle_Home\bpe\samples\tutorials\106.ParallelFlows

**Overview of Parallel Flow Concepts**

Sometimes a BPEL process must gather information from multiple asynchronous sources. Because each callback can take an undefined amount of time (hours or days), it may take too long to call each service one at a time. By breaking the calls into a parallel flow, a BPEL process can invoke multiple Web services at once, and receive the responses as they come in. This method is much more time efficient.

**Figure 6–1** provides an overview of a BPEL process performing a parallel flow to retrieve loan offers from two different Web services. Here, two asynchronous callbacks execute in parallel, so that one callback does not have to wait for the other to complete first. Each response is stored in a different global variable.
Defining a Parallel Flow

A flow activity typically contains a number of sequence activities, and each sequence is performed in parallel. A flow activity can also contain other activities (although not in this example). For example:

```xml
<flow name="flow-1">
  <sequence>
    <scope name="UnitedLoan">
      <sequence>
        <invoke name="invoke-2" partnerLink="unitedLoan" portType="services:LoanService" operation="initiate" inputVariable="loanApplication"/>
        <receive createInstance="no" name="receive-1" partnerLink="unitedLoan" portType="services:LoanServiceCallback" operation="onResult" variable="loanOffer1"/>
      </sequence>
    </scope>
    <sequence>
      <scope name="StarLoan">
        <sequence>
          <invoke name="invoke-1" partnerLink="StarLoan" portType="services:LoanService" operation="initiate" inputVariable="loanApplication"/>
          <pick name="pick-1"/>
        </sequence>
      </scope>
    </sequence>
  </sequence>
</flow>
```

This example shows two sequences, but the flow activity can have many sequences.
The following instructions explain how to create a flow activity and a global input variable named \textit{request}. This activity initiates an asynchronous BPEL process activity with a loan offer Web service (United Loan). The loan offer service uses the request input variable to receive the loan request from the client.

This example shows how to create a flow activity in Oracle JDeveloper.

1. Drag and drop a \textit{flow} activity into a \textit{scope} activity.
2. Click the + sign to expand the \textit{flow} activity.

A \textit{scope} activity is a container for a group of activities that you want to process as one unit. "Using the Scope Activity to Manage a Group of Activities" on page 8-6 describes \textit{scope} activities in detail.

3. The \textit{flow} activity includes two branches, each with a box for functional elements. Populate these boxes as you do a \textit{scope} activity, either by building a function or dragging activities from the Process Activities list into the boxes.

At this point, you can drag and drop activities onto each side of the flow in order to invoke multiple services at once.
Customizing the Number of Flow Activities by Using the flowN Activity

The BPEL code determines the number of parallel branches. However, often the number of branches required is different depending on the available information. The flowN activity creates multiple flows equal to the value of N, which is defined at run time based on the data available and logic within the process. An index variable increments each time a new branch is created, until the index variable reaches the value of N.

The branches created by flowN perform the same activities, but use different data. Each branch uses the index variable to look up input variables. The index variable can be used in the XPath expression to acquire the data specific for that branch.

For example, suppose there is an array of data. The BPEL process uses a count function to determine the number of elements in the array. Then the process sets N to be the number of elements. The index variable starts at a preset value (zero is the default), and flowN creates branches to retrieve each element of the array and perform activities using data contained in that element. These branches are generated and performed in parallel, using all the values between the initial index value and N. flowN terminates when the index variable reaches the value of N. For example, if the array contains 3 elements, N is set to 3. Assuming the index variable begins at 1, the flowN activity creates three parallel branches with indexes 1, 2, and 3.

The flowN activity can use data from other sources as well, including data obtained from Web services.

Figure 6–2 shows an Oracle BPEL Control view of a flowN activity that looks up three hotels. This is different from the view because instead of showing the BPEL process, it shows how the process has actually executed. In this case, there are three hotels, but the number of branches changes to match the number of hotels available.

See Also: The following documentation for examples of creating flow activities in Oracle JDeveloper:
- Oracle BPEL Process Manager Order Booking Tutorial
- Oracle BPEL Process Manager Quick Start Guide
- “Flow Activity” on page B-11
- “Sequence Activity” on page B-24
Customizing the Number of Flow Activities by Using the flowN Activity

Figure 6–2  An Oracle BPEL Control View of the Execution of a flowN activity

Figure 6–3 shows how a flowN activity appears in Oracle JDeveloper.

Figure 6–3  FlowN Activity Setup in the Diagram Window

Figure 6–4 shows the flowN Window, which appears when you double-click the flowN activity.
Customizing the Number of Flow Activities by Using the flowN Activity

Figure 6–4  flowN Window

The flowN window enables you to name the flowN activity, enter an expression for calculating the value of \( N \), and define the index variable.

See Also:  "FlowN Activity" on page B-12

BPEL Code Example of the FlowN Activity

The following code is a reference implementation from a .bpel file that uses the flowN activity to look up information on an arbitrary number of hotels. The following actions take place:

1. First, you name the sequence:

   ```xml
   <sequence name="main">
     <!-- Received input from requestor. Note: This maps to operation defined in NflowHotels.wsdl
     The requestor send a set of hotels names wrapped into the "inputVariable" -->
   </sequence>
   ```

2. The receive activity calls the client partner link to get the information that the flowN activity needs to define \( N \) and look up hotel information:

   ```xml
   <receive name="receiveInput" partnerLink="client" portType="client:NflowHotels" operation="initiate" variable="inputVariable" createInstance="yes"/>
   <!-- The 'count()' Xpath function is used to get the number of hotelName nodes passed in.
       For lissibility, an intermediate variable called "NbParallelFlow" is used to store the number of N flows being executed -->
   <assign name="getHotelsN">
     <copy>
       <from
         expression="count(bpws:getVariableData('inputVariable','payload','/client:NflowHotelsProcessRequest/client:ListOfHotels/client:HotelName'));/>
       <to variable="NbParallelFlow"/>
     </copy>
   </assign>
   <!-- Initiating the FlowN activity
       The N value is initialized with the value stored in the "NbParallelFlow" variable
       The variable call "Index" is defined as the index variable
       NOTE: Both "NbParallelFlow" and "Index" variables have to be declared -->
   ```
The `flowN` activity begins next. After defining a name for the activity of `flowN`, N is defined as a value from the `inputVariable`, which is the number of hotel entries. The activity also assigns index as the index variable.

```bpelx:flowN name="FlowN" N="bpws:getVariableData('NbParallelFlow')" indexVariable="Index">```

```<sequence name="Sequence_1"><!-- Fetching each hotelName by indexing the "inputVariable" with the "Index" variable. Note the usage of the "concat()" Xpath function to create the expression accessing the array element. -->4. Next, the following copy rule uses the index variable to concatenate the hotel entries into a list:```

```<assign name="setHotelId">`<copy>`<from expression="bpws:getVariableData('inputVariable','payload',concat('/client:Nflo wHotelsProcessRequest/client:ListOfHotels/client:HotelName[', bpws:getVariableData('Index'),',']'))"/>`<to variable="InvokeHotelDetailInputVariable" part="payload" query="/ns2:hotelInfoRequest/ns2:id"/></copy>`</assign>`

5. Using the hotel information, an invoke activity looks up detailed information for each hotel through a Web service:

```<!-- For each hotel, invoke the Web service giving detailed information on the hotel --><invoke name="InvokeHotelDetail" partnerLink="getHotelDetail" portType="ns2:getHotelDetail" operation="process" inputVariable="InvokeHotelDetailInputVariable" outputVariable="InvokeHotelDetailOutputVariable"/>`<-- This process doesn't do anything with the retrieved information. In the real life, it could be then used to continue the process. Note: Meanwhile an indexing variable is used, unlike a while loop, the activities are executed in parallel, not sequentially. -->6. Finally, the BPEL process sends detailed information on each hotel to the client partner link:`

```<invoke name="callbackClient" partnerLink="client" portType="client:NfloHotelsCallback" operation="onResult" inputVariable="outputVariable"/>`</invoke>`</sequence>`

3. The `flowN` activity begins next. After defining a name for the activity of `flowN`, N is defined as a value from the `inputVariable`, which is the number of hotel entries. The activity also assigns index as the index variable.
callbacks in parallel, which can take considerably less time than performing the two callbacks in series. Another activity, called a flowN activity, allows Oracle BPEL Process Manager to use data to spawn the necessary number of parallel flows at runtime, and to perform the same activities on multiple data elements. Therefore, as the information available to the BPEL process changes, so does the behavior of the process.
This chapter describes conditional branching. Conditional branching introduces decision points to control the flow of execution of a BPEL process.

This chapter contains the following topics:

- Use Case for Conditional Branching
- Overview of Conditional Branching Concepts
- Using a Switch Activity to Define Conditional Branching
- Using a While Activity to Define Conditional Branching
- Summary

Use Case for Conditional Branching

The BPEL process you created in Chapter 6, "Parallel Flow" collected two loan offers, one from United Loan and another from Star Loan. This chapter describes how to design the BPEL process to select the loan with the lowest annual percentage rate (APR) automatically.

See Also: The following sample:

- [SOA_Oracle_Home\bpel\samples\demos\LoanDemo\LoanFlow](#)

Overview of Conditional Branching Concepts

BPEL applies logic to make choices through conditional branching. You can use either of the following activities to design your code to select different actions based on conditional branching:

- **Switch activity:** In this method, you set up two or more branches, with each branch in the form of an XPath expression. If the expression is true, then the branch is executed. If the expression is false, then the BPEL process moves to the next branch condition, until it either finds a valid branch condition, encounters an otherwise branch, or runs out of branches. If more than one branch condition is true, then BPEL executes the first true branch. "Using a Switch Activity to Define Conditional Branching" on page 7-2 explains how to create switch activities.

- **While activity:** You can use a while activity to create a while loop to select between two actions. "Using a While Activity to Define Conditional Branching" on page 7-4 describes while activities.
A number of branches are set up, and each branch has a condition in the form of an XPath expression.

You can program a conditional branch to have a timeout. That is, if a response cannot be generated in a specified period of time, the BPEL flow can stop waiting and resume its activities. Chapter 10, "Events and Timeouts" explains this feature in detail.

Using a Switch Activity to Define Conditional Branching

In Chapter 6, the flow activity of the BPEL process gathered two loan offers at the same time, but did not compare either of the offers. Each offer was stored in its own global variable. To compare the two offers and make decisions based on that comparison, the BPEL flow requires a switch activity.

Figure 7–1 provides an overview of a BPEL conditional branching process that has been defined in a switch activity.

Figure 7–1 Conditional Branching

A switch activity, like a flow activity, has multiple branches. In this example, there are only two branches. The first branch, which selects a loan offer from United Loan, is executed if a case condition containing an XPath Boolean expression is met. Otherwise, the second branch, which selects the Star Loan loan offer, is executed. By default, the switch activity provides two switch cases, but you can add more if you want.

```xml
<switch name="switch-1">
  <case condition="bpws:getVariableData('loanOffer1','payload','/autoloan:loanOffer/autoloan:APR') <;
   bpws:getVariableData('loanOffer2','payload','/autoloan:loanOffer/autoloan:APR')">
    <assign name="selectUnitedLoan">
      <copy>
        <from variable="loanOffer1" part="payload">
        </from>
        <to variable="selectedLoanOffer" part="payload"/>
      </copy>
    </assign>
  </case>
  <otherwise>
    <assign name="selectStarLoan">
      <switch>
        <case condition="true"/>
        <otherwise>
          <assign>
            Select starLoan
          </assign>
        </otherwise>
      </switch>
    </assign>
  </otherwise>
</switch>
```
Using a Switch Activity to Define Conditional Branching

<assign name="selectStarLoan">
    <copy>
        <from variable="loanOffer2" part="payload"/>
        <to variable="selectedLoanOffer" part="payload"/>
    </copy>
</assign>
</otherwise>
</switch>

Adding a Switch Activity
To add a switch activity to your BPEL flow in Oracle JDeveloper:

1. Drag a switch activity from the Process Activities list of the Component Palette into your BPEL flow.

2. Click the + sign to expand the switch activity.
   The switch activity has two switch case branches by default, each with a box for functional elements. If you want to add more branches, select the entire switch activity, right-click, and select Add Switch Case from the menu.

3. Right-click the first branch and select Edit from the menu.
   The Switch Case window appears.

4. Enter an XPath Boolean expression in the Expression field by pressing the Ctrl key and then the space bar to start the XPath Building Assistant. For example:
   bpws:getVariableDate('loanOffer1','payload','/loanOffer/APR') >
   bpws:getVariableData('loanOffer2','payload','/loanOffer/APR')

5. Enter this expression on one line. To use the XPath Expression Builder, click the XPath Expression Builder icon above the Expression field.
   The two loan offers that the LoanFlow tutorial uses are stored in the global variables loanOffer1 and loanOffer2. Each loan offer variable contains the loan offer’s APR. The BPEL flow must choose the loan with the lower APR. One of the following switch activities takes place:
   - If loanOffer1 has the higher APR, then the first branch selects loanOffer2 by assigning loanOffer2’s payload to selectedLoanOffer’s payload.
Using a While Activity to Define Conditional Branching

- If \( \text{loanOffer1} \) does not have the lower APR than \( \text{loanOffer2} \), then the otherwise case assigns \( \text{loanOffer1} \)'s payload to \( \text{selectedLoanOffer} \)'s payload.

See Also: The following documentation for examples of creating switch activities in Oracle JDeveloper:
- Oracle BPEL Process Manager Order Booking Tutorial
- Oracle BPEL Process Manager Quick Start Guide
- “Switch Activity” on page B-27
- SOA_Oracle_Home\bpe1\samples\references\Switch

Using a While Activity to Define Conditional Branching

Another way to design your BPEL code to select between multiple actions is to use a while activity to create a while loop. The while loop repeats an activity until a specified success criteria is met. For example, if a critical Web service is returning a service busy message in response to requests, you can use the while activity to keep polling the service until it becomes available. The condition for the while activity is that the latest message received from the service is busy, and the operation within the while activity is to check the service again. Once the Web service returns a message other than service busy, the while activity terminates and the BPEL process continues, ideally with a valid response from the Web-service.

To create a while activity in Oracle JDeveloper:

1. Drag and drop a while activity from the Process Activities list of the Component Palette into your BPEL flow.

   The while activity has icons to allow you to build condition expressions and to validate the while definition. It also provides an area for you to drop an activity to define the while loop.

2. Drag the activity that you want to use to define the while condition onto the Drop Activity Here area of the while activity.

   The activity can be an existing activity or a new activity, such as an invoke activity to launch a task.
Summary

This chapter discusses the concepts and procedures for creating a switch activity conditional flow that selects different behavior based on comparing two pieces of information. The BPEL process in this example considers two loan offers, and selects the offer with the lower APR. This chapter also discusses the while looping conditional activity.

See Also: The following documentation for examples of defining a while activity in Oracle JDeveloper:
- “While Activity” on page B-32
- Oracle BPEL Process Manager Order Booking Tutorial
- SOA_Oracle_Home\bpe1\samples\references\While
Fault handling allows a BPEL process to handle error messages or other exceptions returned by outside Web services, and to generate error messages in response to business or run-time faults.

This chapter contains the following topics:

- Use Case for Fault Handling
- Defining a Fault Handler
- BPEL Standard Faults
- Categories of BPEL Faults
- Getting Fault Details with the getFaultAsString XPath Extension Function
- Using the Scope Activity to Manage a Group of Activities
- Throwing Internal Faults
- Returning External Faults
- Using a Fault Handler within a Scope
- Using Compensation After Undoing a Series of Operations
- Using the Terminate Activity to Stop a Business Process Instance
- Fault Handling Example
- Summary

Use Case for Fault Handling

This chapter uses an example of a credit rating service returning a negative credit message instead of a credit rating number. You also learn how to add a fault handler to a BPEL process to handle the message.

See Also: The following samples:

- SOA_Oracle_Home\bpel\samples\tutorials\107.Exceptions
- SOA_Oracle_Home\bpel\samples\demos\ResilientDemo

Defining a Fault Handler

Fault handlers define how the BPEL process responds when the Web services return data other than what is normally expected (for example, returning an error message...
instead of a number). An example of a fault handler is where the Web service normally returns a credit rating number, but instead returns a negative credit message.

Figure 8–1 shows how a fault handler sets the credit rating variable at $-1000$.

The following code segment defines the fault handler for this operation:

```
<faultHandlers>
  <catch faultName="services:NegativeCredit" faultVariable="crError">
    <assign name="crin">
      <copy>
        <from expression="-1000">
        </from>
        <to variable="input" part="payload" query="/autoloan:loanApplication/autoloan:creditRating"/>
      </copy>
    </assign>
  </catch>
</faultHandlers>
```

The `faultHandlers` tag contains the fault handling code. Within the fault handler is a `catch` activity, which defines the fault name and variable, and the `copy` instruction that sets the `creditRating` variable to $-1000$.

When you select Web services for the BPEL process, determine the possible faults that may be returned and set up a fault handler for each one.
BPEL Standard Faults

The Business Process Execution Language for Web Services Specification defines the following standard faults in the namespace of http://schemas.xmlsoap.org/ws/2003/03/business-process/:

- selectionFailure
- conflictingReceive
- conflictingRequest
- mismatchedAssignmentFailure
- joinFailure
- forcedTermination
- correlationViolation
- uninitializedVariable
- repeatedCompensation
- invalidReply

Standard faults are defined as follows:
- Typeless, meaning they do not have associated messageTypes
- Not associated with any WSDL message
- Caught without a fault variable:
  <catch faultName="bpws:selectionFault">

Categories of BPEL Faults

A BPEL fault has a fault name called a QName (name qualified with a namespace) and a possible messageType. There are two categories of BPEL faults:

- Business faults
- Run-time faults

Business Faults

Business faults are application-specific faults that are generated when there is a problem with the information being processed (for example when a social security number is not found in the database). A business fault occurs when an application executes a throw activity or when an invoke activity receives a fault as a response. The fault name of a business fault is specified by the BPEL process. The messageType, if applicable, is defined in the WSDL. A business fault can be caught with a faultHandler using the faultName and a faultVariable.

<catch faultName="ns1:FaultName" faultVariable="varName"/>

Run-time Faults

Run-time faults are the result of problems within the running of the BPEL process or Web service (for example, data cannot be copied properly because the variable name is incorrect). These faults are not user-defined, and are thrown by the system. They are generated if the process tries to use a value incorrectly, a logic error occurs (such as an
endless loop), a SOAP fault occurs in a SOAP call, an exception is thrown by Oracle BPEL Server, and so on.

Oracle BPEL Server includes several run-time faults. These faults are included in the http://schemas.oracle.com/bpel/extension namespace. These faults are associated with the messageType RuntimeFaultMessage. The following WSDL file defines the messageType:

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<definitions name="RuntimeFault"
    targetNamespace="http://schemas.oracle.com/bpel/extension"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns="http://schemas.xmlsoap.org/wsdl/">
    <message name="RuntimeFaultMessage">
        <part name="code" type="xsd:string" />
        <part name="summary" type="xsd:string" />
        <part name="detail" type="xsd:string" />
    </message>
</definitions>
```

If a faultVariable (of messageType RuntimeFaultMessage) is used when catching the fault, the fault code can be queried from the faultVariable, along with the fault summary and detail.

**bindingFault**

A bindingFault is thrown inside an activity if the preparation of the invocation fails. For example, the WSDL of the process fails to load. A bindingFault is not retryable. This type of fault usually must be fixed by human intervention. Table 8-1 describes the fault codes.

<table>
<thead>
<tr>
<th>Fault Code</th>
<th>Description of Fault</th>
</tr>
</thead>
<tbody>
<tr>
<td>VersionMismatch</td>
<td>The processing party found an invalid namespace for the SOAP envelope element.</td>
</tr>
<tr>
<td>MustUnderstand</td>
<td>An immediate child element of the SOAP header element that was either not understood or not obeyed by the processing party contained a SOAP MustUnderstand attribute with a value of 1.</td>
</tr>
<tr>
<td>Client_GenericError</td>
<td>Generic error on the client side</td>
</tr>
<tr>
<td>Client_WrongNumberOfInputParts</td>
<td>Input message part number mismatch</td>
</tr>
<tr>
<td>Client_WrongNumberOfOutputParts</td>
<td>Output message part number mismatch</td>
</tr>
<tr>
<td>Client_WrongTypeOfInputPart</td>
<td>Input message part type error</td>
</tr>
<tr>
<td>Client_WrongTypeOfOutputPart</td>
<td>Output message part type error</td>
</tr>
<tr>
<td>Server_GenericError</td>
<td>Generic error on the server side</td>
</tr>
<tr>
<td>Server_NoService</td>
<td>Server is up, but there is no service</td>
</tr>
<tr>
<td>Server_NoHTTPSOAPAction</td>
<td>Request is missing the HTTP SOAP action</td>
</tr>
<tr>
<td>Server_Unauthenticated</td>
<td>Request is not authenticated</td>
</tr>
<tr>
<td>Server_Unauthorized</td>
<td>Request is not authorized</td>
</tr>
</tbody>
</table>
remoteFault

A remoteFault is also thrown inside an activity. It is thrown because the invocation fails. For example, a SOAP fault is returned by the remote service. A remoteFault can be configured to be retried. Table 8–2 describes the fault codes.

Table 8–2 remoteFault Fault Codes

<table>
<thead>
<tr>
<th>Fault Code</th>
<th>Description of Fault</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConnectionRefused</td>
<td>Remote server is unavailable</td>
</tr>
<tr>
<td>WSDLReadingError</td>
<td>Failed to read the WSDL</td>
</tr>
<tr>
<td>GenericRemoteFault</td>
<td>Generic remote fault</td>
</tr>
</tbody>
</table>

replayFault

A replayFault replays the activity inside a scope. At any point inside a scope, this fault is migrated up to the scope. Oracle BPEL Server then re-executes the scope from the beginning.

Catching Run-time Faults Example

BPEL run-time faults can be caught as a named BPEL fault. The bindingFault and remoteFault can be associated with a message. This enables the faultHandler to get details about the faults.

The following procedure shows how to use the provided examples to generate a fault and define a fault handler to catch it. In this case, you modify a WSDL file to generate a fault, and create a catch attribute to catch it.

1. Import RuntimeFault.wsdl into your process WSDL (located under the SOA_Oracl...e_bpel\system\xmllib directory).
2. Declare a variable with messageType bpelx:RuntimeFaultMessage.
3. Catch it using

   `<catch faultName="bpelx:remoteFault" | "bpelx:bindingFault"
    faultName="varName">`

See Also: The following sample, which describes how to handle run-time binding faults:

- SOA_Oracl...e_bpel\samples\demos\ResilientDemo

Getting Fault Details with the getFaultAsString XPath Extension Function

The catchAll activity is provided to catch possible faults. However, BPEL does not provide a method for obtaining additional information about the captured fault. Use the getFaultAsString() XPath extension function to obtain additional information.

```
<catchAll>
  <sequence>
    <assign>
      <from expression="bpelx:getFaultAsString()"/>
      <to variable="faultVar" part="message"/>
    </assign>
    <reply faultName="ns1:myFault" variable="faultVar" .../>
  </sequence>
</catchAll>
```
Using the Scope Activity to Manage a Group of Activities

The scope activity provides a container and a context for other activities. A scope provides handlers for faults, events, and compensation, as well as data variables and correlation sets. Using a scope activity simplifies a BPEL flow by grouping functional structures together. This allows you to collapse them into what appears to be a single element in Oracle JDeveloper.

The following code example shows a scope activity. In this case, the process for getting a credit rating based on a customer’s social security number has been placed inside a scope named `getCreditRating`. This identifies functional blocks of code and sets them apart visually. In Oracle JDeveloper, you can collapse the activities contained inside the scope into a single visual element, or expand them when necessary.

```xml
<scope name="getCreditRating">
  <variables>
    <variable name="crError" messageType="services:CreditRatingServiceFaultMessage"/>
  </variables>
  <assign name="assign-2">
    <copy>
      <to variable="input" part="payload" query="/autoloan:loanApplication/autoloan:creditRating"/>
    </copy>
  </assign>
</sequence>
</scope>
```

To add a scope activity:

1. Click and drag a scope activity into the BPEL process diagram.
2. Open the scope by double-clicking it or by single-clicking the + sign.
3. Drag activities from the Component Palette to build the function within the scope.

See Also: The following documentation for examples of creating scope activities in Oracle JDeveloper:
- Oracle BPEL Process Manager Order Booking Tutorial
- Oracle BPEL Process Manager Quick Start Guide
- “Scope Activity” on page B-22

Throwing Internal Faults

A BPEL application can generate and receive fault messages. The throw activity has three elements: its name, the name of the fault:Name, and the fault:Variable. If you add a throw activity to your BPEL process, it automatically includes a copy rule that copies the fault name and type into the output payload. The fault thrown by a throw activity is internal to BPEL. You cannot use a throw activity on an asynchronous process to communicate with a client. Here is a code sample of a throw activity, which includes the fault elements, name, and partner link of the service to which the BPEL process sends the fault, and the copy rule that packages the message:

```xml
<throw name="delay" faultName="fault-1" faultVariable="fVar"/>
<invoke name="invokesStockQuoteService" partnerLink="StockQuoteService"/>
<assign>
  <copy>
    <from variable="response" part="result" query="/result"/>
    <to variable="output" part="payload" query="/tns:result"/>
  </copy>
</assign>
```
Returning External Faults

A BPEL process can send a fault to another application to indicate a problem, as opposed to throwing an internal fault. In a synchronous operation, the reply activity can return the fault. In an asynchronous operation, the invoke activity performs this function.

Returning a Fault in a Synchronous Interaction

The syntax of a reply activity that returns a fault in a synchronous interaction is as follows:

```xml
<reply partnerLink="partner-link-name" portType="port-type-name" operation="operation-name" variable="variable-name" (optional) faultName="fault-name">
</reply>
```

Always returning a fault in response to a synchronous request is not very useful. It is better to make the activity part of a conditional branch, where the first branch is executed if the data requested is available. If the requested data is not available, then the BPEL process returns a fault with this information.

See Also:
- Chapter 7, “Conditional Branching” for more information on setting up the conditional structure
- Chapter 4, “Invoking a Synchronous Web Service” for more information on synchronous interactions
- “Reply Activity” on page B-21

Returning a Fault in an Asynchronous Interaction

In an asynchronous interaction, the client does not wait for a reply. The reply activity is not used to return a fault. Instead, the BPEL process returns a fault using a callback operation on the same port type that normally receives the requested information, with an invoke activity.

See Also:
- Chapter 5, “Invoking an Asynchronous Web Service” for more information on asynchronous interactions
- “Invoke Activity” on page B-14
Using a Fault Handler within a Scope

If a fault is not handled, it creates a faulted state that migrates up through the application and can throw the entire process into a faulted state. To prevent this, contain the parts of the process that have the potential to receive faults within a scope. As described earlier, the scope activity includes fault handling capabilities. The catch activity works within a scope to catch faults and exceptions before they can throw the entire process into a faulted state.

You can use specific fault names in the catch activity to respond in a specific way to an individual fault. To catch any faults that are not already handled by name-specific catch activities, use the catchAll activity.

See Also: The following documentation for examples of creating fault handling:
- Oracle BPEL Process Manager Order Booking Tutorial
- “Scope Activity” on page B-22
- SOA_Oracle_Home\bpel\samples\references\Catch

Using the Empty Activity to Insert No-Op Instructions into a Business Process

There is often a need to use an activity that does nothing. An example is when a fault must be caught and suppressed. In this case, you can use the empty activity to insert a no-op instruction into a business process. The syntax to use an empty activity is as follows:

```xml
<empty standard-attributes>
  standard-elements
</empty>
```

If no catch or catchAll is selected, the fault is not caught by the current scope and is rethrown to the immediately enclosing scope. If the fault occurs in (or is rethrown to) the global process scope, and there is no matching fault handler for the fault at the global level, the process terminates abnormally. This is as though a terminate activity (described in “Using the Terminate Activity to Stop a Business Process Instance” on page 8-10) had been performed.

Consider the following example:

```xml
<faultHandlers>
  <catch faultName="x:foo">
    <empty/>
  </catch>
  <catch faultVariable="bar">
    <empty/>
  </catch>
  <catch faultName="x:foo" faultVariable="bar">
    <empty/>
  </catch>
  <catchAll>
    <empty/>
  </catchAll>
</faultHandlers>
```

Assume that a fault named x:foo is thrown. The first catch is selected if the fault carries no fault data. If there is fault data associated with the fault, the third catch is selected if the type of the fault’s data matches the type of variable bar. Otherwise, the default catchAll handler is selected. Finally, a fault with a fault variable whose type
Using Compensation After Undoing a Series of Operations

Compensation occurs when the BPEL process cannot complete a series of operations after some of them have already completed, and the BPEL process must backtrack and undo the previously completed transactions. For example, if a BPEL process is designed to book a rental car, a hotel, and a flight, it may book the car and the hotel and then be unable to book a flight for the right day. In this case, the BPEL flow performs compensation by going back and unbooking the car and the hotel.

You can invoke a compensation handler by using the compensate activity, which names the scope for which the compensation is to be performed (that is, the scope whose compensation handler is to be invoked). A compensation handler for a scope is available for invocation only when the scope completes normally. Invoking a compensation handler that has not been installed is equivalent to using the empty activity (it is a no-op). This ensures that fault handlers do not have to rely on state to determine which nested scopes have completed successfully. The semantics of a process in which an installed compensation handler is invoked more than once are undefined.

If an invoke activity has a compensation handler defined inline, then the name of the activity is the name of the scope to be used in the compensate activity. The syntax is as follows:

```xml
<compensate scope="ncname" standard-attributes>
  standard-elements
</compensate>
```

The ability to explicitly invoke the compensate activity is the underpinning of the application-controlled error-handling framework of Business Process Execution Language for Web Services Specification. You can use this activity only in the following parts of a business process:

- In a fault handler of the scope that immediately encloses the scope for which compensation is to be performed.
- In the compensation handler of the scope that immediately encloses the scope for which compensation is to be performed.

For example:

```xml
<compensate scope="RecordPayment"/>
```

If a scope being compensated by name was nested in a loop, the BPEL process invokes the instances of the compensation handlers in the successive iterations in reverse order.

If the compensation handler for a scope is absent, the default compensation handler invokes the compensation handlers for the immediately enclosed scopes in the reverse order of the completion of those scopes.

The compensate form, in which the scope name is omitted in a compensate activity, explicitly invokes this default behavior. This is useful when an enclosing fault or

See Also:
- “Scope Activity” on page B-22
- “Empty Activity” on page B-8
compensation handler must perform additional work, such as updating variables or sending external notifications, in addition to performing default compensation for inner scopes. The compensate activity in a fault or compensation handler attached to the outer scope invokes the default order of compensation handlers for completed scopes directly nested within the outer scope. You can mix this activity with any other user-specified behavior except for the explicit invocation of the nested scope within the outer scope. Explicitly invoking a compensation for such a scope nested within the outer scope disables the availability of default-order compensation.

See Also:
- "BankTransferDemo" on page 1-7 for a demonstration that uses a compensate activity
- "Compensate Activity" on page B-4

Using the Terminate Activity to Stop a Business Process Instance

The terminate activity immediately terminates the behavior of a business process instance within which the terminate activity is performed. All currently running activities must be terminated as soon as possible without any fault handling or compensation behavior. The terminate activity does not send any notifications of the status of a BPEL process. If you are going to use the terminate activity, first program notifications to the interested parties.

The syntax for the `terminate` activity is as follows:

```xml
<terminate standard-attributes>
  standard-elements
</terminate>
```

See Also: The following documentation for examples of creating terminate activities:
- "Terminate Activity" on page B-27
- `SOA_Oracle_Home\bpel\samples\references\Terminate`

Fault Handling Example

The ResilientDemo sample demonstrates failover fault handling and retry fault handling. Failover allows multiple service implementations to be configured for a partner link. If a retryable run-time fault occurs, then the server tries other service implementations. In retry fault handling, the server retries based on a specified retry interval and retry count. Another kind of fault, a binding fault, can occur if the Web service has been upgraded and the interface has changed. In the ResilientDemo sample, when a binding fault occurs, the document is placed in a dead letter queue using a JMS service. The diagram of ResilientFlow.bpmi is shown in Figure 8–2.
The invokeRatingService activity shows the failover feature. The partner link of this invoke has two possible implementations, which are configured in the deployment descriptor file as follows:

```xml
<properties id="RatingService">
  <property name="wsdlLocation">
    http://localhost:8080/axis/services/RatingService1?wsdl
  </property>
  <property name="location">
    http://localhost:1234/axis/services/RatingService1
    http://localhost:8080/axis/services/RatingService2
  </property>
</properties>
```

The preceding code sample shows that two endpoint locations are configured for the RatingService partner link. The first endpoint is a bad URL and the second endpoint is a good URL. Because a remote exception like this is retryable, and there is a second endpoint, Oracle BPEL Server tries to call the second endpoint, at which point, the call succeeds.

The invokeFlakyService activity, expanded in Figure 8–3, shows system retry.
The partner link of this invoke is configured as follows:

```
<properties id="FlakyService">
  <property name="wsdlLocation">
    http://localhost:8080/axis/services/FlakyService?wsdl</property>
  <property name="location">
    http://localhost:2222/axis/services/FlakyService</property>
  <property name="retryMaxCount">2</property>
  <property name="retryInterval">60</property>
</properties>
```

If the service is not listening on port 2222, then the invoke fails with a ConnectionRefused run-time fault. Because this is a retryable fault, and the retryMaxCount (set to 2) and retryInterval parameters (set to 60) are defined, Oracle BPEL Server retries twice, with 60 second intervals between each attempt. The second retry is successful.

**See Also:** The following sample:

- SOA_Oracle_Home\bpel\samples\demos\ResilientDemo
- SOA_Oracle_Home\bpel\samples\demos\ResilientDemo\ResilientFlow\ResilientFlow.pdf for instructions

**Summary**

BPEL supports fault handlers to cope with faults, errors, or exceptions returned by the called Web services. This chapter demonstrates the application of a fault handler, a fault handler’s structure, and how to create a fault handler in a BPEL process.
Incorporating Java and J2EE Code in BPEL Processes

You can embed sections of Java code into a BPEL process. This chapter contains the following topics:

- Overview of Java and J2EE Code in BPEL Concepts
- Using Java Embedding in a BPEL Process
- Summary

Overview of Java and J2EE Code in BPEL Concepts

This chapter explains how you can embed sections of Java code into a BPEL process. This is particularly useful when there is already Java code that can perform the desired function, and you want to use the existing code rather than start over with BPEL.

You can incorporate Java code using any of the following methods:

- Using Java Code with WSIF Binding
- Using Java Code Wrapped as a SOAP Service
- Directly Embedding Java Code in a BPEL Process

Using Java Code with WSIF Binding

If the Java application has a BPEL-compatible interface, you either use Web Services Inspection Language (WSIF) binding or wrap the Java code as a SOAP service to use it in a BPEL process.

WSIF binding is the most common way of using Java code in a BPEL process. This method enables a BPEL process to invoke an Enterprise Java Bean through native J2EE protocol (local or remote method invocation (RMI)). With WSIF binding, a section of the WSDL file defines the protocol for communicating between Java and XML. This approach maintains Java’s transactionality and does not sacrifice performance. It is also quicker for you to add WSIF binding to an existing Java application rather than starting over in Oracle BPEL Process Manager. However, WSIF binding has the following drawbacks:

- Less tool support than SOAP services
- Less interoperability, because each application server needs a specific binding

Currently, you must write the binding manually.
Oracle BPEL Process Manager’s Java binding implementation is based on the Apache WSIF package’s Java binding. Java binding enables a BPEL process to invoke user-defined Java classes.

All data used internally in a BPEL process is of a W3C DOM element. Therefore, data marshalling must be performed between a BPEL process and the user-defined Java classes. Oracle BPEL Process Manager’s Java binding implementation currently supports Java classes that use the following data types:

- XML simple type
- XML complex type using Oracle BPEL Process Manager facade data type
- DOM element

### Java Binding Service Using XML Simple Types

Most XML simple types can be mapped to Java types, and vice versa. The data mapping can be defined in the WSDL file using the format extension. For example, a Java method is defined as follows:

```java
public float getQuote (String symbol) throws Exception;
```

The `format:typeMap` definitions are defined as follows:

```xml
<format:typeMapping encoding="Java" style="Java">
  <format:typeMap typeName="xsd:string" formatType="java.lang.String" />
  <format:typeMap typeName="xsd:float" formatType="java.lang.float" />
</format:typeMapping>
```

### Java Binding Service Using Oracle BPEL Process Manager XML Facade

An XML facade is an Oracle BPEL Process Manager technology that provides a Java bean-like interface on top of an XML DOM element. Given the XML schemas, facade classes can be generated using the Oracle BPEL Process Manager `schemagen` tool. In the following example, the classes use XML facade classes in the method:

```java
public CommentsType addComment(CommentsType payload, CommentType comment) throws JavaBindingException;
```

The WSDL Java binding is defined as follows:

```xml
<format:typeMapping encoding="Java" style="Java">
  <format:typeMap type="tns:commentType" formatType="com.otn.services.CommentType" />
  <format:typeMap type="tns:commentsType" formatType="com.otn.services.CommentsType" />
</format:typeMapping>
```

See Also: [SOA Oracle Home](samples\demos\IBMSamples\simple) for a sample of using XML simple types

### Java Binding Service Using Oracle BPEL Process Manager XML Facade

An XML facade is an Oracle BPEL Process Manager technology that provides a Java bean-like interface on top of an XML DOM element. Given the XML schemas, facade classes can be generated using the Oracle BPEL Process Manager `schemagen` tool. In the following example, the classes use XML facade classes in the method:

```java
public CommentsType addComment(CommentsType payload, CommentType comment) throws JavaBindingException;
```

The WSDL Java binding is defined as follows:

```xml
<format:typeMapping encoding="Java" style="Java">
  <format:typeMap type="tns:commentType" formatType="com.otn.services.CommentType" />
  <format:typeMap type="tns:commentsType" formatType="com.otn.services.CommentsType" />
</format:typeMapping>
```
Overview of Java and J2EE Code in BPEL Concepts

Incorporating Java and J2EE Code in BPEL Processes

```xml
<operation name="addComment">
  <java:operation methodName="addComment"/>
  <input/>
  <output/>
  <fault name="CommentException"/>
</operation>
<operation name="testFault">
  <java:operation methodName="testFault"/>
  <input/>
  <output/>
  <fault name="CommentException"/>
</operation>
</binding>

The Java types `CommentType` and `CommentsType` are XML facade classes.

**See Also:**
- "Using an XML Facade to Simplify DOM Manipulation" on page 9-7
- "schemac" on page 19-29 for details about the schemac tool and XML facades

### Java Binding Service Using XML DOM Elements

Oracle BPEL Process Manager internally uses an XML DOM element. If a Java binding class uses XML DOM elements, data marshalling is not needed. For example, in the following example, the `addComment`() method is defined:

```java
public Element addComment(Element payload, Element comment)
  throws JavaBindingException;
```

The Java binding of the WSDL file is defined as follows:

```xml
<binding name="JavaBinding2" type="tns:HelperService2">
  <java:binding/>
  <format:typeMapping encoding="Java" style="Java">
    <format:typeMap typeName="tns:commentType" formatType="org.w3c.dom.Element"/>
    <format:typeMap typeName="tns:commentsType" formatType="org.w3c.dom.Element"/>
  </format:typeMapping>
  <operation name="addComment">
    <java:operation methodName="addComment"/>
    <input/>
    <output/>
    <fault name="CommentException"/>
  </operation>
  <operation name="testFault">
    <java:operation methodName="testFault"/>
    <input/>
    <output/>
    <fault name="CommentException"/>
  </operation>
</binding>
```

With the Java class using a DOM element type, Java binding can support any XML data type.

Incorporating Java and J2EE Code in BPEL Processes 9-3
Throwing a WSDL Fault from a Java Binding

As a Web service, a Java binding can throw a fault. The fault can be defined in the WSDL file as a Web service:

```xml
<message name="CommentFaultMessage">
  <part name="payload" type="tns:commentType"/>
</message>

<portType name="HelperService2">
  <operation name="testFault">
    <input message="tns:TestFaultRequestMessage"/>
    <output message="tns:TestFaultResponseMessage"/>
    <fault name="CommentException" message="tns:CommentFaultMessage" />
  </operation>
</portType>

<binding name="JavaBinding2" type="tns:HelperService2">
  <operation name="testFault">
    <java:operation methodName="testFault"/>
    <input/>
    <output/>
    <fault name="CommentException" />
  </operation>
</binding>
```

To throw a Web service fault, the Java class must declare to throw a predefined fault type `com.collaxa.cube.ws.wsif.providers.java.JavaBindingException` or its subclass.

In the following example, the Java file has a function that throws this exception:

```java
public CommentType testFault(CommentType payload)
    throws JavaBindingException
{
    System.out.println("testFault : " + payload);
    if (payload != null) {
        JavaBindingException ex = new JavaBindingException();
        ex.setFaultName("CommentException");
        ex.setPart("payload", payload);
        System.out.println(" throwing exception " + ex);
        throw ex;
    } else {
        return payload;
    }
}
```

In a different Java file, there is a similar function:

```java
public Element testFault(Element payload)
    throws JavaBindingException
{
    System.out.println("testFault2: " + payload);
    if (payload != null) {
        JavaBindingException ex = new JavaBindingException();
        ex.setFaultName("CommentException");
        ex.setPart("payload", payload);
        System.out.println(" throwing exception " + ex);
        throw ex;
    } else {
        return payload;
    }
}
```
Overview of Java and J2EE Code in BPEL Concepts

Incorporating Java and J2EE Code in BPEL Processes

9-5

If such an exception is thrown from the Java binding class, the BPEL WSIF layer converts it into a Web service exception that can be caught by the BPEL source as follows:

```xml
<variables>
    ........
    <variable name="fault" messageType="services:CommentFaultMessage"/>
</variables>

<faultHandlers>
    ........
    <catch faultName="services:CommentException" faultVariable="fault">
        <assign>
            <copy>
                <from expression="string('CommentException')"/>
                <to variable="output" part="payload" query="/types:comments/types:item[1]/types:message"/>
            </copy>
        </assign>
    </catch>
    <catchAll>
        <empty/>
    </catchAll>
</faultHandlers>
```

See Also:
- Chapter 8, “Fault Handling” for additional details about faults
- SOA_Oracle_Home\bpel\samples\tutorials\702.Bindings for examples of WSIF bindings for EJB, HTTP, and Java. Bindings must be written to match the application server.
- SOA_Oracle_Home\bpel\samples\demos\BankTransferDemo\BankTransferFlow

Using Java Code Wrapped as a SOAP Service

As an alternative to WSIF binding, you can wrap the Java code as a SOAP service. As with WSIF binding, this method requires that the Java application have a BPEL-compatible interface. A Java application wrapped as a SOAP service appears as any other Web service, which can be used by many different kinds of applications. There are also tools available for writing SOAP wrappers.

However, a Java application wrapped as a SOAP service has the following drawbacks:
- It loses performance, because interactions are constantly being mapped back and forth between the Java code and the SOAP wrapper.
- It loses interoperability, that is, the ability to perform several operations in an all-or-none mode (such as debiting one bank account while crediting another, where either both transactions must be completed, or neither of them).
Directly Embedding Java Code in a BPEL Process

Another way to use Java in a BPEL process is to embed the code directly into the BPEL process using the Java BPEL exec extension `bpelx:exec`. The benefits of this approach are speed and transactionality. However, you can incorporate only fairly small segments of code. If you want to incorporate larger segments of code, or if the project requires the Java code to have the same look and feel throughout all the BPEL processes being created, consider using WSIF binding or wrapping it as a SOAP service.

Using the `bpelx:exec` Tag to Embed Java Code Snippets into a BPEL Process

The BPEL tag `bpelx:exec` enables you to embed a snippet of Java code within a BPEL process. The server executes any snippet of Java code contained within a `bpelx:exec` activity, within its Java Transaction API (JTA) transaction context.

The BPEL tag `bpelx:exec` converts Java exceptions into BPEL faults and then adds them into the BPEL process.

The Java snippet can propagate its JTA transaction to session and entity beans that it calls.

For example, a `SessionBeanSample.bpel` file uses the following `bpelx:exec` tag to embed the `invokeSessionBean` Java bean:

```xml
<bpel:exec name="invokeSessionBean" language="java" version="1.4">
  <![CDATA[
    try {
      Object homeObj = lookup("ejb/session/CreditRating");
      Class cls = Class.forName("com.otn.samples.sessionbean.CreditRatingServiceHome");
      CreditRatingServiceHome ratingHome = (CreditRatingServiceHome)
        PortableRemoteObject.narrow(homeObj,cls);
      if (ratingHome == null) {
        addAuditTrailEntry("Failed to lookup 'ejb.session.CreditRating'
          + '. Please make sure that the bean has been
          + ' successfully deployed'");
        return;
      }
      CreditRatingService ratingService = ratingHome.create();
      // Retrieve ssn from scope
      Element ssn = (Element)getVariableData("input","payload","/ssn");
      int rating = ratingService.getRating( ssn.getNodeValue() );
      addAuditTrailEntry("Rating is: " + rating);
      setVariableData("output", "payload", "/tns:rating", new Integer(rating));
    } catch (NamingException ne) {
      addAuditTrailEntry(ne);
    } catch (ClassNotFoundException cnfe) {
      addAuditTrailEntry(cnfe);
    } catch (CreateException ce) {
      addAuditTrailEntry(ce);
    } catch (RemoteException re) {
      addAuditTrailEntry(re);
    }
  ]]></bpel:exec>
```
Using an XML Facade to Simplify DOM Manipulation

You can use an XML facade to simplify DOM manipulation. Oracle BPEL Process Manager provides a lightweight Java Architecture for XML Binding (JAXB)-like Java object model on top of XML (called a facade). An XML facade provides a Java bean-like front end for an XML document or element that has a schema. Facade classes can provide easy manipulation of the XML document and element in Java programs.

You add the XML facade by using a createFacade method within the bpelx:exec statement in the .bpel file. For example:

```xml
<bpelx:exec name="...">
  <![CDATA[
    (Element)getVariableData("input","payload","/loanApplication/"):
    //Create an XMLFacade for the Loan Application Document
    LoanApplication xmlLoanApp=
      LoanApplicationFactory.createFacade(element);
  ]]>...
</bpelx:exec>
```

To generate the facade classes, use the schemac tool, which is provided with Oracle BPEL Process Manager. You can find the schemac tool in the following locations:

- `SOA_Oracle_Home/bpel/bin`

To use schemac, run a command similar to the following to generate the facades from WSDL or XSD files:

```
C:\BPEL_project_dir\> schemac *.wsdl /*.xsd
```

After you run schemac, it creates a `src` folder for a `HelperService.java` service and a `com` folder for the generated Java classes. Oracle provides a sample in the following directories that showcases the use of facade classes in Java bindings:

- `SOA_Oracle_Home/bpel/samples/tutorials/702.Bindings/JavaBinding`

When it generates the facade, schemac uses the following files:

- Using `build.xml`, schemac generates the source of the facade classes.
- The schemac tool creates a Java binding provider class `HelperService.java`, which in the 702.Bindings example is located under `702.Bindings/JavaBinding/src/com/otn/services`. It has one method, which uses the facade classes `CommentsType` and `CommentType`:

  ```java
default public CommentsType addComment(CommentsType payload, CommentType comment)
```

- To map the XML types to the corresponding facade classes, a Java binding service is defined in the `HelperService.wsdl` file. See the `format:typeMapping` section of Java binding:

```xml
  <format:typeMapping encoding="Java" style="Java">
    <format:typeMap typeName="tns:commentType" formatType="com.otn.services.CommentType" />
    <format:typeMap typeName="tns:commentsType" formatType="com.otn.services.CommentsType" />
  </format:typeMapping>
```

See Also: "schemac" on page 19-29
Using Java Embedding in a BPEL Process

Table 9–1 lists a set of bpelx:exec built-in methods that you can use to read and update scope variables, instance metadata, and audit trails.

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object lookup( String name )</td>
<td>JNDI access</td>
</tr>
<tr>
<td>Locator getLocator( )</td>
<td>BPEL Process Manager Locator</td>
</tr>
<tr>
<td>long getInstanceId( )</td>
<td>Unique ID associated with each instance</td>
</tr>
<tr>
<td>String setTitle( String title ) /</td>
<td>Title of this instance</td>
</tr>
<tr>
<td>String getTitle()</td>
<td></td>
</tr>
<tr>
<td>String setStatus( String status ) /</td>
<td>Status of this instance</td>
</tr>
<tr>
<td>String getStatus()</td>
<td></td>
</tr>
<tr>
<td>void setIndex( int i, String value ) / String getIndex( int i )</td>
<td>Six indexes can be used for search</td>
</tr>
<tr>
<td>void setPriority( int priority ) /</td>
<td>Priority</td>
</tr>
<tr>
<td>int getPriority()</td>
<td></td>
</tr>
<tr>
<td>void setCreator( String creator ) / String getCreator( )</td>
<td>Who initiated this instance</td>
</tr>
<tr>
<td>void setCustomKey( String customKey ) / String getCustomKey( )</td>
<td>Second primary key</td>
</tr>
<tr>
<td>void setMetadata( String metadata ) / String getMetadata( )</td>
<td>Metadata for generating lists</td>
</tr>
<tr>
<td>String getPreference( String key )</td>
<td>Access preference defined in bpel.xml</td>
</tr>
<tr>
<td>void addAuditTrailEntry( String message, Object detail)</td>
<td>Add an entry to the audit trail</td>
</tr>
<tr>
<td>void addAuditTrailEntry( Throwable t)</td>
<td>Add an entry to the audit trail with exception</td>
</tr>
<tr>
<td>Object getVariableData( String name ) throws BPELFault</td>
<td>Access and update variables stored in the scope</td>
</tr>
<tr>
<td>Object getVariableData( String name, String partOrQuery) throws BPELFault</td>
<td></td>
</tr>
<tr>
<td>Object getVariableData( String name, String part, String query)</td>
<td></td>
</tr>
<tr>
<td>void setVariableData( String name, Object value)</td>
<td></td>
</tr>
<tr>
<td>void setVariableData( String name, String part, Object value)</td>
<td></td>
</tr>
<tr>
<td>void setVariableData( String name, String part, String query, Object value)</td>
<td></td>
</tr>
</tbody>
</table>

Using Java Embedding in a BPEL Process

In Oracle JDeveloper, you can add the bpelx:exec activity, and copy the code snippet into a dialog box, as follows:

1. Drag and drop the Java Embedding activity (with the coffee cup icon) from the Component Palette.
2. Double-click the Java Embedding activity to display the Java Embedding window.
3. Name the Java Embedding activity.

4. In the Code Snippet field, enter (or cut and paste) the Java code.

For example, the `<bpel:exec>` code example described under “Using the `<bpelx:exec>` Tag to Embed Java Code Snippets into a BPEL Process” on page 9-6 appears as follows:

```java
try {
    Object bpeXml = [Java code here];
    Class ext = [Java code here];
    Code snippet CodeSnippetName = [Java code here];
    PostProcessSubstitutionContextContext=>$[[$cmd]]($context);($[[$cmd]]($context));($context);
    if (resultFlag == true) {
        // Add code here
    }
}
```

See Also: “Java Embedding Activity” on page B-16 for additional details about this activity, including adding JAR files to classpaths.

Summary

This chapter demonstrates how you can embed sections of Java code into a BPEL process using one of the following techniques:

- If the Java application has a BPEL-compatible interface, you can use WSIF binding or wrap the Java code in a SOAP service.
- You can directly embed the Java code by including an inline code snippet using `<bpelx:exec>`. This snippet is executed within the transaction context of Oracle BPEL Server. This method allows you to propagate that transaction to your own session and entity beans. You can use a set of built-in methods to enable the `<bpelx:exec>` snippet to read and update variables, change instance meta data, and throw faults. To simplify DOM manipulation, use an XML facade.
This chapter describes how to use events and timeouts. Because Web services can take a long time to return a response, a BPEL process must be able to time out and continue with the rest of the flow after a period of time.

This chapter contains the following topics:

- Use Case for Events and Timeouts
- Overview of Event and Timeout Concepts
- Using the Pick Activity to Select Between Continuing a Process or Waiting
- Using the Wait Activity to Set an Expiration Time
- Setting Timeouts for Synchronous Processes
- Defining a Timeout
- Summary

Use Case for Events and Timeouts

In this use case, you program a BPEL process to wait one minute for a response from the Star Loan Web service. If Star Loan does not respond in one minute, then the BPEL process automatically selects the United Loan offer. In the real world, the time limit is more like 48 hours. However, for this example, you do not want to wait that long to see if your BPEL process is working properly.

See Also: The following sample file:

- SOA_Oracle_Home\bpel\samples\tutorials\108.Timeouts

Overview of Event and Timeout Concepts

Because asynchronous Web services can take a long time to return a response, a BPEL process must be able to time out, or give up waiting, and continue with the rest of the flow after a certain amount of time. You can use the pick activity to configure a BPEL flow to either wait a specified amount of time or to continue performing its duties. To set an expiration period for the time, you can use the wait activity.

If you plan to set timeouts for synchronous processes that connect to a remote database, you need to set the syncMaxWaitTime timeout property in the domain.xml file.
Using the Pick Activity to Select Between Continuing a Process or Waiting

The pick activity provides two branches, each one with a condition. The branch that has its condition satisfied first is executed. In the following example, one branch’s condition is to receive a loan offer, and the other branch’s condition is to wait a specified amount of time.

Figure 10-1 provides an overview. The following activities take place:

1. An invoke activity initiates a service, in this case, a request for a loan offer from Star Loan.

2. The pick activity begins next. It has the following conditions:
   - **onMessage**: This condition has code for receiving a reply in the form of a loan offer from the Star Loan Web service. The onMessage code is the same as the code for receiving a response from the Star Loan Web service before a timeout was added.
   - **onAlarm**: This condition has code for a timeout of one minute. This time is defined as PT1M, which means to wait one minute before timing out. In this timeout setting, S stands for seconds, H for hour, D for day, and Y for year. In the unlikely event that you want a time limit of 1 year, 3 days, and 15 seconds, you enter it as PT1Y3D15S. The remainder of the code sets the loan variables selected and approved to false, sets the annual percentage rate (APR) at 0.0, and copies this information into the loanOffer variable.

   For more detailed information on the time duration format, see the duration section of the most current XML Schema Part 2: Datatypes document at: http://www.w3.org/TR/xmlschema-2/#duration

3. The pick activity condition that completes first is the one that the BPEL process executes. The other branch then is not executed.
Using the Pick Activity to Select Between Continuing a Process or Waiting

**Figure 10–1 Overview of the Pick Activity**

The following code segment defines the `pick` activity for this operation:

```xml
<pick>
  <!-- receive the result of the remote process -->
  <onMessage partnerLink="LoanService"
             portType="services:LoanServiceCallback"
             operation="onResult" variable="loanOffer">
    <assign>
      <copy>
        <from variable="loanOffer" part="payload"/>
        <to variable="output" part="payload"/>
      </copy>
    </assign>
  </onMessage>
  <!-- wait for one minute, then timeout -->
  <onAlarm for="PT1M">
    <assign>
      <copy>
        <from>
          <loanOffer xmlns="http://www.autoloan.com/ns/autoloan">
            <providerName>Expired</providerName>
            <selected type="boolean">false</selected>
            <approved type="boolean">false</approved>
            <APR type="double">0.0</APR>
          </loanOffer>
        </from>
        <to variable="loanOffer" part="payload"/>
      </copy>
    </assign>
  </onAlarm>
</pick>
```

Events and Timeouts 10-3
Using the Wait Activity to Set an Expiration Time

The wait activity allows a process to wait for a given time period or until a time limit has been reached. Exactly one of the expiration criteria must be specified.

```xml
<wait (for="duration-expr" | until="deadline-expr") standard-attributes>
    standard-elements
</wait>
```

**See Also:** The following documentation for examples of defining a wait activity:
- "SleepBroker" on page 1-8 for a demonstration that uses a wait activity
- "Wait Activity" on page B-31
- Oracle BPEL Process Manager Order Booking Tutorial
- SOA_Oracle_Home\bpel\samples\references\Wait

Setting Timeouts for Synchronous Processes

For synchronous processes that connect to a remote database, you must increase the syncMaxWaitTime timeout property in the SOA_Oracle_Home\bpel\domains\default\config\domain.xml file:

```xml
<property id="syncMaxWaitTime">
    <name>Delivery result receiver maximum wait time</name>
    <value>45</value>
    <comment><![CDATA[The maximum time the process result receiver will wait for a result before returning. Results from asynchronous BPEL processes are retrieved synchronously via a receiver that will wait for a result from the container. The default value is 45 seconds.]]>
    </comment>
</property>
```

**Defining a Timeout**

To define a timeout, follow these steps:

1. Drag and drop a Pick activity into a BPEL process.
   - The Pick activity includes the onMessage (envelope icon) and onAlarm (alarm clock icon) branches.
2. Double-click the OnAlarm branch of the onAlarm activity and set its time limit to 1 minute instead of 1 hour.

3. Click OK.

4. Double-click the onMessage activity, and edit its attributes to receive the response from the loan service.
Instead of performing multiple operations at the same time as with the flow attribute, you can use the pick activity to define a number of operations such that only the first one to complete is executed. The example in this chapter is of a pick activity where one branch is an asynchronous callback from a loan service, and the other branch is a timeout set at one minute.

See Also: “Pick Activity” on page B-19

Summary

Instead of performing multiple operations at the same time as with the flow attribute, you can use the pick activity to define a number of operations such that only the first one to complete is executed. The example in this chapter is of a pick activity where one branch is an asynchronous callback from a loan service, and the other branch is a timeout set at one minute.
This chapter shows how a Java or JSP application can call a BPEL process to perform functions or use services.

This chapter contains the following topics:
- Use Case for Invoking a BPEL Process
- Overview of Invoking BPEL Process Concepts
- Sending Messages to a BPEL Process from a Java or JSP Application
- Summary

Use Case for Invoking a BPEL Process

In this use case, you learn how to invoke synchronous and asynchronous BPEL processes through either the simple object access protocol (SOAP) or Java. The BPEL process accepts a social security number and sends a credit rating in return. The user Web interface is provided by a JSP file, which takes the input and passes it to a BPEL process to get back a credit rating.

See Also: The following sample file:
- SOA_Oracle_Home\bpe1\samples\tutorials\102.InvokingProcesses

Overview of Invoking BPEL Process Concepts

A Java or a JSP application can call a BPEL process to perform functions or use services. A BPEL process is itself a Web service, defining and supporting a client interface through WSDL and SOAP. However, you can make BPEL processes deployed on Oracle BPEL Process Manager available to clients through a Java API.


Sending Messages to a BPEL Process from a Java or JSP Application

You can invoke a BPEL process as a Web service through a WSDL or SOAP interface, or as a Java component through its client Java interface. The application puts the request in the form of a payload that then goes to the BPEL process. The BPEL process receives the payload and responds with a payload containing the information that the application requested.
Invoking a BPEL Process with the Generic Java API

You can invoke a BPEL process by using a generic Java API. Oracle provides classes that enable your BPEL process to use a generic Java API to connect to Oracle BPEL Process Manager and to pass XML messages through a generic Java API. You can use these classes to perform either two-way or one-way invoke operations.

This section covers the following topics:

■ Connecting to Oracle BPEL Process Manager with the Locator Class
■ Passing XML Messages Through Java
■ Invoking a Two-Way Operation Through the Java API
■ Invoking a One-Way Operation Through the Java API

Connecting to Oracle BPEL Process Manager with the Locator Class

Oracle provides a `com.oracle.bpel.client.Locator` class that supports a flexible client interface without being affected by server clustering and other production details. Use this class to do the following:

■ Connect to Oracle BPEL Process Manager, authenticating if required
■ Obtain handles to services provided by Oracle BPEL Server

For example, the `Locator` class can connect to the default domain on a local Oracle BPEL Process Manager and fetch a list of BPEL processes deployed on that server. In this case, the `Locator` class returns a handle to a `com.oracle.bpel.client.dispatch.IDeliveryService` instance.

The following instance can invoke or initiate BPEL processes on Oracle BPEL Server:

```java
import com.oracle.bpel.client.Locator;
import com.oracle.bpel.client.dispatch.IDeliveryService;

// Connect to domain "default" using password "bpel"
// null IP address means local server
```
Invoking a BPEL Process

```
Locator locator = new Locator("default", "welcome1", null);
IDeliveryService deliveryService = (IDeliveryService)locator.lookupService
    (IDeliveryService.SERVICE_NAME);
```

Passing XML Messages Through Java
Because all Web services, including BPEL processes, accept and return XML messages,
you can use the Oracle BPEL Process Manager client class
com.oracle.bpel.client.NormalizedMessage to activate an XML message
dynamically.

For example, to activate an input message for the CreditRatingService from static
string XML data, you can use the following code:
```
import com.oracle.bpel.client.NormalizedMessage;
String xml = "<ssn xmlns="http://services.otn.com">123456789</ssn>";
NormalizedMessage nm = new NormalizedMessage();
    nm.addPart("payload", xml);
```

Invoking a Two-Way Operation Through the Java API
After a delivery service has been instantiated, it can initiate a BPEL process with a
NormalizedMessage XML message. You can use one of the
IDeliveryService.request() methods to invoke a two-way Web service
operation, which has an input message and returns a result synchronously.
The IDeliveryService.request() method is overloaded. To find out more about
its available versions, refer to the Oracle BPEL Process Manager Javadoc.
In this version, the request() method has the following signature:
```
public NormalizedMessage request(java.lang.String processId,
    java.lang.String operationName,
    NormalizedMessage message)
    throws java.rmi.RemoteException
```

The following code example (provided with the Oracle BPEL Process Manager
samples) demonstrates how to use this API to invoke the CreditRatingService
BPEL process.
```
<%@page import="java.util.Map" %>
<%@page import="com.oracle.bpel.client.Locator" %>
<%@page import="com.oracle.bpel.client.NormalizedMessage" %>
<%@page import="com.oracle.bpel.client.dispatch.IDeliveryService" %>
<html>
    ...
```
Sending Messages to a BPEL Process from a Java or JSP Application

```java
String ssn = request.getParameter("ssn");
if(ssn == null)
    ssn = "123-12-1234";
String xml = "<ssn xmlns="http://services.otn.com">" + ssn + "</ssn>";
Locator locator = new Locator("default", "bpel", null);
IDeliveryService deliveryService =
    (IDeliveryService)locator.lookupService
    (IDeliveryService.SERVICE_NAME);
// construct the normalized message and send to oracle bpel process manager
NormalizedMessage nm = new NormalizedMessage();
nm.addPart("payload", xml);
NormalizedMessage res =
    deliveryService.request("CreditRatingService", "process", nm);
Map payload = res.getPayload();
out.println("BPELProcess CreditRatingService executed!\n");
out.println("Credit Rating is " + payload.get("payload")
);
```

**See Also:** The following sample:
- SOA_Oracle_
  - Home\bpe\samples\tutorials\102.InvokingProcesses
  - jsp\invokeCreditRatingService.jsp

Invoking a One-Way Operation Through the Java API

A one-way invoke operation has only an input message and does not return a result. The procedure for invoking a one-way BPEL operation through the Java API is very similar to how you invoke two-way operations. The difference is that you use the `IDeliveryService.post()` method instead of `IDeliveryService.request()`. This method is overloaded; its methods invoke a one-way operation on a BPEL process and thus return `void` because a response is not expected (at least not a synchronous response).

From the Javadoc for `com.oracle.bpel.client.dispatch.IDeliveryService`:

```java
public void post(java.lang.String processId,
    java.lang.String operationName,
    NormalizedMessage message)
    throws java.rmi.RemoteException
```

In the following example, the `post()` method is very similar to the `request()` method shown in the two-way example discussed earlier, except that it returns `void`.

```java
<%@page import="com.oracle.bpel.client.Locator" %>
<%@page import="com.oracle.bpel.client NORMALIZEDMESSAGE" %>
<%@page import="com.oracle.bpel.client.dispatch.IDeliveryService" %>
...
Locator locator = new Locator("default", "bpel", null);
...
NormalizedMessage nm = new NormalizedMessage();
    nm.addPart("payload", xml);
deliveryService.post("HelloWorld", "initiate", nm);
```

11-4 Oracle BPEL Process Manager Developer's Guide
Sending Messages to a BPEL Process from a Java or JSP Application

Invoking a BPEL Process

11-5

out.println("BPELProcess HelloWorld initiated!");
%

See Also: The following sample:
- SOA_Oracle/Home/bpel/samples/tutorials/102.InvokingProcesses/jsp/invokeHelloWorld.jsp

Retrieving Status or Results from Asynchronous BPEL Processes

If you use the Java API to initiate an asynchronous BPEL process, you must often consider how to receive the result of the process, because a typical Java client cannot be called back the same way as a Web service. You can handle this problem by using the following strategies:

- Have your code inform users of the progress of the process. For example, the LoanFlowPlus BPEL demonstration application (located in SOA_Oracle/Home/bpel/samples/demos/LoanDemoPlus) informs users of the progress through a user task, such as manually approving the final loan offer. You also can have the process send some sort of notification, such as an e-mail message or a JMS message, when it completes.

- For asynchronous BPEL processes, have the Java client poll the result. In this case, the client needs a handle to fetch status information for a particular instance. The post() method does not automatically return such a handle, but it does allow the client to specify a conversation ID. This ID can be any unique identifier that the client can later use to identify a specific instance and retrieve status information for it. See the Oracle BPEL Javadocs for the com.oracle.bpel.client.NormalizedMessage class to find the specific field name for the conversation ID and other properties, which you can set at the time a BPEL process is instantiated through the Java API. You can also use the com.oracle.bpel.client.Locator.lookupInstance(String key) method to locate a specific instance based on a conversation ID.

You also can use the NormalizedMessage properties to specify the address of a Web service for the callback. This initiates an asynchronous BPEL process from Java, but receives a SOAP/XML callback to a Web service listener.

It is also possible using the supported NormalizedMessage properties to specify the address of a Web service for the callback and therefore initiate an asynchronous BPEL process from Java, but receive a SOAP/XML callback to a Web service listener.

Contact Oracle Support Services for more information on how to retrieve status or results from an asynchronous BPEL process in your specific environment.

Using the Java API from a Remote Client

The code examples described in previous sections are executed within the same application server container in which the Oracle BPEL Process Manager is running. You can run these APIs from a remote client, however, and use them through a remote method invocation (RMI) from a remote application server. The RMI client code you use depends on the application server in which the client is running. Work with Oracle Support Services regarding how to use the Oracle BPEL Process Manager Java API over RMI for your specific client configuration and environment.
Invoking a BPEL Process with the Web Service/SOAP Interface

After you deploy a BPEL process to Oracle BPEL Server, it is automatically published as a Web service. This means that the process can be accessed through its XML/SOAP/WSDL interface without any additional developer effort. Supporting a standard Web services interface means that BPEL processes can be invoked from any client technology that supports Web services. This includes Microsoft .NET, Sun’s JAX-RPC implementation, Apache Axis, Oracle JDeveloper, and many other Web services tool kits. In addition, it means that BPEL and Oracle BPEL Process Manager can publish Web services. Those services, both synchronous and asynchronous, can be invoked from applications and services implemented with nearly any technology and language.

You access a BPEL process through its Web service interface in the standard way you access any Web service: by writing a client that uses the BPEL process WSDL interface definition and SOAP as a protocol.

Summary

Once deployed, a BPEL process is exposed through both a WSDL or SOAP interface and a business delegate Java interface. The Java business delegate interface allows Java and JSP applications to initiate new instances of a BPEL process.

The Java business delegate can be used locally or remotely using RMI. The Java business delegate is JTA-aware, allowing the initiation of a process to be part of a broader transaction.
This chapter identifies common interaction patterns between a BPEL process and another application, and shows the best use practices for each.

This chapter contains the following topics:

- One-Way Message
- Synchronous Interaction
- Asynchronous Interaction
- Asynchronous Interaction with Timeout
- Asynchronous Interaction with a Notification Timer
- One Request, Multiple Responses
- One Request, One of Two Possible Responses
- One Request, a Mandatory Response, and an Optional Response
- Partial Processing
- Multiple Application Interactions
- Summary

**One-Way Message**

In a one-way message, or fire and forget, the client sends a message to the service, and the service does not need to reply. Figure 12-1 provides an overview.
Synchronous Interaction

**Figure 12–1 One-Way Message**

As the client, the BPEL process needs a valid partner link and an invoke activity with the target service and the message. As with all partner activities, the WSDL file defines the interaction.

**BPEL Process as the Client**
As the client, the BPEL process needs a valid partner link and an invoke activity with the target service and the message. As with all partner activities, the WSDL file defines the interaction.

**BPEL Process as the Service**
To accept a message from the client, the BPEL process needs a receive activity.

**Synchronous Interaction**

In a synchronous interaction, a client sends a request to a service, and receives an immediate reply. The BPEL process can be at either end of this interaction, and must be coded based on its role as either the client or the service. Figure 12–2 provides an overview.
Asynchronous Interaction

BPEL Process as the Client
When the BPEL process is on the client side of a synchronous transaction, it needs an invoke activity. The port on the client side both sends the request and receives the reply. As with all partner activities, the WSDL file defines the interaction.

BPEL Process as the Service
When the BPEL process is on the service side of a synchronous transaction, it needs a receive activity to accept the incoming request, and a reply activity to return either the requested information or an error message (a fault).

See Also: Chapter 4, “Invoking a Synchronous Web Service”

Asynchronous Interaction
In an asynchronous interaction, a client sends a request to a service and waits until the service replies. Figure 12–3 provides an overview.
Asynchronous Interaction with Timeout

Figure 12–3 Asynchronous Interaction

BPEL Process as the Client
When the BPEL process is on the client side of an asynchronous transaction, it needs an invoke activity to send the request and a receive activity to receive the reply. As with all partner activities, the WSDL file defines the interaction.

BPEL Process as the Service
As with a synchronous transaction, when the BPEL process is on the service side of an asynchronous transaction, it needs a receive activity to accept the incoming request and an invoke activity to return either the requested information or a fault.

See Also: Chapter 5, "Invoking an Asynchronous Web Service"

Asynchronous Interaction with Timeout
In an asynchronous interaction with a timeout, a client sends a request to a service and waits until it receives a reply, or until a certain time limit is reached, whichever comes first. Figure 12–4 provides an overview.
Asynchronous Interaction with a Notification Timer

**Interaction Patterns**

**Figure 12–4** Asynchronous Interaction with Timeout

When the BPEL process is on the client side of an asynchronous transaction with a timeout, it needs an invoke activity to send the request and a pick activity with two branches: an onMessage branch and an onAlarm branch. If the reply comes after the time limit has expired, the message goes to the dead letter queue. As with all partner activities, the WSDL file defines the interaction.

**BPEL Process as the Client**

When the BPEL process is on the client side of an asynchronous transaction with a timeout, it needs an invoke activity to send the request and a pick activity with two branches: an onMessage branch and an onAlarm branch. If the reply comes after the time limit has expired, the message goes to the dead letter queue. As with all partner activities, the WSDL file defines the interaction.

**See Also:** "Using the Pick Activity to Select Between Continuing a Process or Waiting" on page 10-2

**BPEL Process as the Service**

The behavior of the service BPEL process is the same as with the asynchronous interaction with the BPEL process as the service, as described in "BPEL Process as the Service" on page 12-4.

**Asynchronous Interaction with a Notification Timer**

In an asynchronous interaction with a notification time, a client sends a request to a service and waits for a reply, although a notification is sent after a timer expires. The client continues to wait for the reply from the service even after the timer has expired. Figure 12–5 provides an overview.

**Figure 12–5** Asynchronous Interaction with a Notification Timer

- **Call service**
- **onMessage**
- **Logic**
- **Post**
- **Callback**
- **Time out in 1M**
- **onAlarm**
- **Logic**
- **Post**
- **Timeout**

**Deployment Descriptor**

*(bpel.xml)*

-WSDL*

-PartnerLink
d1
d2

**Call**

**Service BPEL Process**

- **receive**
- **invoke**

**Client BPEL Process**

- **pick**
- **Wait for callback**
- **onMessage**

**Extra Text:**

- Time out in 1M
- onAlarm
- Logic
- Post
- Callback
- Time out in 1M
- onAlarm
- Logic
- Post
- Timeout

**Diagram:**

- **Deployment Descriptor**
- **WSDL**
- **PartnerLink**
- **d1**
- **d2**
- **Call service**
- **onMessage**
- **Logic**
- **Post**
- **Callback**
- **Time out in 1M**
- **onAlarm**
- **Logic**
- **Post**
- **Timeout**

**Interaction Patterns** 12-5
One Request, Multiple Responses

BPEL Process as the Client
When the BPEL process is on the client side of this transaction, it needs a scope activity containing an invoke activity to send the request, and a receive activity to accept the reply. The onAlarm handler of the scope activity has a time limit and instructions on what to do when the timer expires. For example, wait 30 minutes, then send a warning indicating that the process is taking longer than expected. As with all partner activities, the WSDL file defines the interaction.

BPEL Process as the Service
The behavior for the service BPEL process is the same as with the asynchronous interaction with the BPEL process as the service, as described in “BPEL Process as the Service” on page 12-4.

One Request, Multiple Responses
In this interaction type, the client sends a single request to a service and receives multiple responses in return. For example, the request can be to order a product online, and the first response can be the estimated delivery time, the second response a payment confirmation, and the third response a notification that the product has shipped. In this example, the number and types of responses are expected. Figure 12-6 provides an overview.
BPEL Process as the Client
When the BPEL process is on the client side of this transaction, it needs an invoke activity to send the request, and a sequence activity with three receive activities, one for each reply. As with all partner activities, the WSDL file defines the interaction.

BPEL Process as the Service
The BPEL service needs a receive activity to accept the message from the client, and a sequence attribute with three invoke activities, one for each reply.

One Request, One of Two Possible Responses
In an interaction using one request and one of two possible responses, the client sends a single request to a service and receives one of two possible responses. For example, the request can be to order a product online, and the first response can be either an in-stock message, or an out-of-stock message. Figure 12-7 provides an overview.
One Request, a Mandatory Response, and an Optional Response

When the BPEL process is on the client side of this transaction, it needs the following:

- An invoke activity to send the request
- A pick activity with two branches: one onMessage for the in-stock response and instructions on what to do if an in-stock message is received
- A second onMessage for the out-of-stock response and instructions on what to do if an out-of-stock message is received

As with all partner activities, the WSDL file defines the interaction.

**See Also:** “Using the Pick Activity to Select Between Continuing a Process or Waiting” on page 10-2

BPEL Process as the Service

The BPEL service needs a receive activity to accept the message from the client, and a switch activity with two branches, one with an invoke activity sending the in-stock message if the item is available, and a second branch with an invoke activity sending the out-of-stock message if the item is not available.

One Request, a Mandatory Response, and an Optional Response

In this type of interaction, the client sends a single request to a service and receives one or two responses. Here, the request is to order a product online. If the product is delayed, the service sends a message letting the customer know. In any case, the service always sends a notification when the item ships. Figure 12–8 provides an overview.
BPEL Process as the Client
When the BPEL process is on the client side of this transaction, it needs a scope activity containing the invoke activity to send the request, and a receive activity to accept the mandatory reply. The onMessage handler of the scope activity is set to accept the optional message and instructions on what to do if the optional message is received (for example, notify you that the product has been delayed). The client BPEL process waits to receive the mandatory reply. If the mandatory reply is received first, the BPEL process continues without waiting for the optional reply. As with all partner activities, the WSDL file defines the interaction.

BPEL Process as the Service
The BPEL service needs a scope activity containing the receive activity and an invoke activity to send the mandatory shipping message, and the scope’s onAlarm handler to send the optional delayed message if a timer expires (for example, send the delayed message if the item is not shipped in 24 hours).

Partial Processing
In partial processing, the client sends a request to a service and receives an immediate response, but processing continues on the service side. For example, the client sends a request to purchase a vacation package, and the service sends an immediate reply.
confirming the purchase, then continues on to book the hotel, the flight, the rental car, and so on. This pattern can also include multiple shot callbacks, followed by longer-term processing. Figure 12–9 provides an overview.

**Figure 12–9 Partial Processing**

BPEL Process as the Client

In this case, the BPEL client is simple; it needs an invoke activity for each request and a receive activity for each reply for asynchronous transactions, or just an invoke activity for each synchronous transaction. Once those transactions are complete, the remaining work is handled by the service. As with all partner activities, the WSDL file defines the interaction.

BPEL Process as the Service

The BPEL service needs a receive activity for each request from the client, and a reply activity for each response. Once the responses are finished, the service BPEL process can continue with its processing, using the information gathered in the interaction to perform the necessary tasks without any further input from the client.

Multiple Application Interactions

In some cases, there are more than two applications involved in a transaction, for example, a buyer, seller, and shipper. In this case, the buyer sends a request to the seller, the seller sends a request to the shipper, and the shipper sends a notification to the buyer. This A-to-B-to-C-to-A transaction pattern can handle many transactions at once. Therefore, a mechanism is required for keeping track of which message goes where. Figure 12–10 provides an overview.

As with all partner activities, the WSDL file defines the interaction.
BPEL processes can serve as both clients or services, and this chapter lists several common interaction patterns and describes best practices for implementing these interactions.
This part describes how Oracle BPEL Process Manager adds value and ease of use to key BPEL development concepts to support the following services.

This part contains the following chapters:

- Chapter 13, “XSLT Mapper and Transformations”
- Chapter 14, “Oracle BPEL Process Manager Notification Service”
- Chapter 15, “Oracle BPEL Process Manager Workflow Services”
- Chapter 16, “Worklist Application”
- Chapter 17, “Sensors”
- Chapter 18, “BPEL Process Integration with Business Rules”
This chapter describes features of the XSLT Mapper and provides step-by-step instructions for mapping a sample purchase order schema to an invoice schema.

This chapter contains the following topics:

- Use Case for Transformation
- Creating an XSL Map File
- Overview of the XSLT Mapper
- Using the XSLT Mapper
- Testing the Map
- Summary

Use Case for Transformation

Transformation use is demonstrated in several Oracle BPEL Process Manager use cases.

See:

- SOA_Oracle_home\bpel\samples\demos\XSLMapper
- SOA_Oracle_home\bpel\samples\tutorials\114.XSLTTransformations
- Oracle BPEL Process Manager Order Booking Tutorial

Creating an XSL Map File

Transformations are performed in an XSL map file in which you map source schema elements to target schema elements. This section describes two methods for creating the XSL map file:

- Creating a New XSL Map File
- Creating an XSL Map File from Imported Source and Target Schema Files

Note: You can also create an XSL map file from an XSL stylesheet. Click New > General > XML > XSL Map From XSL Stylesheet from the File main menu in Oracle JDeveloper.
Creating an XSL Map File

Creating a New XSL Map File

A transform activity enables you to create a transformation using the XSLT Mapper tool. This tool enables you to map source elements to target elements. For example, you can map incoming source purchase order schema data to outgoing invoice schema data.

1. Drag and drop a transform activity from the Component Palette into your BPEL process diagram.

2. Double-click the transform activity.
   The Transform window appears.

3. Specify the following information:
   - Source variable from which to map elements
   - Source part of the variable (for example, a payload schema consisting of a purchase order request) from which to map
   - Target variable to which to map elements
Creating an XSL Map File

Target part of the variable (for example, a payload schema consisting of an invoice) to which to map.

4. Specify a map file name or accept the default name in the Mapper File field. The map file is the file in which you create your mappings using the XSLT Mapper transformation tool.

5. Click the magic wand icon (second icon) to create a new mapping. If the file already exists, click the note pad icon (third icon) to edit the mapping. The XSLT Mapper appears.

6. Go to "Overview of the XSLT Mapper" on page 13-5 for an overview of using the XSLT Mapper.

Creating an XSL Map File from Imported Source and Target Schema Files

**Note:** If you select a file with a .xslt extension such as xform.xslt, it opens the mapper pane to create a new XSL file named xform.xslt.xsl, even though your intention was to use the existing xform.xslt file. A .xsl extension is appended to any file that does not already have a .xsl extension, and you must create the mappings in the new file. As a workaround, ensure that your files first have an extension of .xsl. If the XSL file has an extension of .xslt, then rename it to .xsl.

The following steps provide a high level overview of how to create an XSL map using the existing po.xsd and invoice.xsd files in the SOA_Oracle_home\pe1\samples\demos\XSLMapper directory.

1. In Oracle JDeveloper, select the application project in which you want to create the new XSL map.

2. Import the po.xsd and invoice.xsd files into the project (for example, by right-clicking Schemas and selecting Import Schemas in the Structure section of Oracle JDeveloper).

3. Right-click the selected project and select New.

   The New Gallery window appears.

4. In the Categories tree, expand General and select XML.

5. In the Items list, double-click XSL Map.

   The Create XSL Map File window appears. This window enables you to create an XSL map file that maps a root element of a source schema file or WSDL file to a root element of a target schema file or WSDL file.

   - Schema files that have been added to the project appear under Project Schema Files.
   - Schema files that are not part of the project can be imported using the Import Schema File facility. Click the Import Schema File icon (first icon to the right and above the list of schema files).

6. Enter a name for the XSL map file in the File Name field.

7. Under Source, expand Project Schema Files > po.xsd > PurchaseOrder as the root element for the source.
Creating an XSL Map File

8. Under Target, expand Project Schema files > invoice.xsd > Invoice as the root element for the target.

9. Click OK.
   A new XSL map is created.

10. Save and close the file now or begin to design your transformation. Information on using the XSLT Mapper tool is provided in “Overview of the XSLT Mapper” on page 13-5.

11. Drag and drop a transform activity from the Component Palette into your BPEL process.

12. Double-click the transform activity.

13. Specify the following information:
   - Source variable from which to map elements
Source part of the variable (for example, a payload schema consisting of a purchase order request) from which to map
- Target variable to which to map elements
- Target part of the variable (for example, a payload schema consisting of an invoice) to which to map

14. Click the flashlight icon (first icon) to the right of the Mapper File field to browse for the map file name you specified in Step 6.
15. Click Open.
16. Click OK.

The XSLT Mapper displays your XSL map file.

17. Go to "Overview of the XSLT Mapper" on page 13-5 for an overview of using the XSLT Mapper.

**Overview of the XSLT Mapper**

You use the XSLT Mapper transformation tool to create the contents of a map file. Figure 13-1 shows the layout of the XSLT Mapper.

![Figure 13-1 Layout of the XSLT Mapper](image)

The **Source** and the **Target** schemas are represented as trees and the nodes in the trees are represented using a variety of icons. The displayed icon reflects the schema or property of the node. For example:

- An XSD attribute is denoted with an icon that is different from an XSD element
- An optional element is represented with an icon that is different from a mandatory element
Using the XSLT Mapper

- A repeating element is represented with an icon that is different from a nonrepeating element, and so on

The various properties of the element and attribute are displayed in the Property Inspector in the lower right of Figure 13–1 (for example, type, cardinality, and so on). The Functions Palette in the upper right of Figure 13–1 is the container for all functions provided by the XSLT Mapper. The mapper pane or canvas is the actual drawing area for dropping functions and connecting them to source and target nodes.

The XSLT Mapper provides three separate context sensitive menus:
- One in the source panel
- One in the target panel
- One in the mapper pane or canvas in the middle

Right-click each of the three separate panels to see what the context menus look like. A full set of Undo Auto Map, Redo, Delete, and Delete All functions are also available.

Notes on the Mapper

- A node in the target tree can be linked only once (that is, you cannot have two links connecting a node in the target tree).
- An incomplete function and expression does not result in an XPath expression in the source view. If you switch from the design view to the source view with one or more incomplete expressions, the Mapper Messages window displays warning messages.
- When you map duplicate elements in the XSLT Mapper, the style sheet becomes invalid and you cannot work in the Design view. The Log Window shows the following error messages when you map an element with a duplicate name:

```
Error: This Node is Already Mapped : 
"/ns0:rulebase/for-each/ns0:if/ns0:atom/ns0:rel"
Error: This Node is Already Mapped :
"/ns0:rulebase/for-each/ns0:if/ns0:atom/choice_1/ns0:ind"
Error: This Node is Already Mapped :
"/ns0:rulebase/for-each/ns0:if/ns0:atom/choice_1/ns0:var"
```

The workaround is to give each element a unique name.

Using the XSLT Mapper

The following sections describe how to use the XSLT Mapper:

- Simple Copy by Linking Nodes
- Setting Constant Values
- Adding Functions
- Editing XPath Expressions
- Adding XSLT Constructs
- Automatically Mapping Nodes
- Viewing Unmapped Target Nodes
- Generating Dictionaries
- Creating Map Parameters and Variables
Using the XSLT Mapper

- Searching Source and Target Nodes
- Ignoring Elements in the XSLT Document
- Replacing a Schema in the XSLT Mapper

Simple Copy by Linking Nodes

To copy an attribute or leaf-element in the source to an attribute or leaf-element in the target, drag and drop the source to the target. Copy the element `PurchaseOrder/ID` to `Invoice/ID` and the attribute `PurchaseOrder/OrderDate` to `Invoice/InvoiceDate`, as shown in Figure 13–2.

Figure 13–2 Linking Nodes

Setting Constant Values

Perform the following steps to set a constant value.

1. Select a node in the target tree.
2. Invoke the context menu by right-clicking the mouse.
3. Select the Set Text menu option.
4. Enter text in the Set Text window (for example, Discount Applied, as shown in Figure 13–3).
5. Click OK to save the text.
   
   A T icon is displayed next to the node that has text associated with it.
6. If you want to remove the text associated with the node, right click the node to invoke the Set Text window again. Delete the contents and click OK.

Figure 13–3 Set Text Window
Adding Functions

In addition to the standard XPath 1.0 functions, the Mapper provides a number of prebuilt extension functions and has the ability to support user-defined functions and named templates. The extension functions are prefixed with `xp20` or `orcl` and mimic XPath 2.0 functions.

Perform the following steps to view function definitions and use a function:

1. Select a category of functions (for example, String Functions) from the Component Palette.
2. Right-click an individual function (for example, lower-case).
3. Select Help. A window with a description of the function appears. You can also click a link at the bottom to access this function's description at the World Wide Web Consortium at www.w3.org.
4. Drag a concat function into the mapper pane. This function enables you to connect the source parameters from the source tree to the function and the output of the function to the node on the target tree.
5. Concatenate PurchaseOrder/ShipTo/Name/First and PurchaseOrder/ShipTo/Name/Last. Place the result in Invoice/ShippedTo/Name by dragging threads from the first and last names and dropping them on the left side on the concat function. Also drag a thread from the ShippedTo name and connect it to the right side on the concat function, as shown in Figure 13-4.

See Also: The online Help for the Set Text window for detailed information
Using the XSLT Mapper

**Figure 13–4 Using the Concat Function**

See Also: The documentation for the XPath extension functions, which is described in Appendix D, "XPath Extension Functions"

**Editing Function Parameters**

To edit the parameters of the `concat` function, double-click the function icon to launch the Edit Function - `concat` window. This window enables you to add, remove, and reorder parameters. If you want to add a new comma parameter so that the output of the `concat` function is `Last, First`, then click **Add** to add a comma and reorder the parameters to get this output.

**Figure 13–5 Editing Function Parameters**
Using the XSLT Mapper

Chaining Functions

Complex expressions can be built by chaining functions. To remove all leading and trailing spaces from the output of the above `concat` function, use the `left-trim` and `right-trim` functions and chain them as shown in the Figure 13–6.

The chaining function can also be defined by dragging and dropping the function to a connecting link.

Figure 13–6  Chaining Functions

Named Templates

Some complicated mapping logic cannot be represented or achieved by visual mappings. For these situations, named templates are useful. Named templates enable you to share common mapping logic. You can define the common mapping logic as a named template and then use it as often as you want.

You define named templates in the source view, and they appear in the User Defined Named Templates list of the Component Palette. You can use named templates in almost the same way as you use other functions. The only difference is that you cannot link the output of a named template to a function or another named template; you can only link its output to a target node in the target tree.

To write named templates, you must be familiar with the XSLT language. See any XSLT book or visit the following URL for details about writing named templates:

http://www.w3.org/TR/xslt

Importing User-Defined Functions

You can import your own set of Java functions, which appear in the function palette under the User Defined Extension Functions category. They can be used like any other function. To add functions, select Preferences > XSL Maps from the Tools main menu.

See Also:  SOA_Oracle_home\bpel\samples\demos\XSLMapper\ExtensionFunctions\README.txt for detailed instructions
Editing XPath Expressions

To use an XPath expression in a transformation mapping, select Advanced Functions from the Component Palette and drag and drop xpath-expression from the list into the transformation window, as shown in Figure 13–7.

When you double-click the icon, the Edit XPath Expression window appears, as shown in Figure 13–8. You can press the Ctrl key and then the spacebar to invoke the XPath Building Assistant.

Figure 13–9 shows the XPath Building Assistant.
Using the XSLT Mapper

Figure 13–9 The XPath Building Assistant

Adding XSLT Constructs

While mapping complex schemas, it is sometimes essential to conditionally map a source node to a target or map an array of elements in the source to an array of elements in the target. The XSLT Mapper provides various XSLT constructs in the context sensitive menu of the target tree for the preceding scenarios. To add an XSLT element like `for-each`, `if`, or `choose` to a schema element, select the element in the target tree. Right-click and select Add XSL Node to bring up the context menu and choose the required XSLT element in the menu.

Conditional Processing with `xsl:if`

Note that HQAccount and BranchAccount are part of a choice in the PurchaseOrder schema; only one of them exists in an actual instance. To illustrate conditional mapping, copy `PurchaseOrder/HQAccount/AccountNumber` to `Invoice/BilledToAccount/AccountNumber` only if it exists. To do this:

1. Select `Invoice/BilledToAccount/AccountNumber` in the target tree and right-click to bring up the context sensitive menu.
2. Select Add XSL Node, and then if and connect `PurchaseOrder/HQAccount/AccountNumber` to `Invoice/BilledToAccount/if/AccountNumber`.
3. Connect `PurchaseOrder/HQAccount/AccountNumber` to `Invoice/BilledToAccount/if/AccountNumber`.

Figure 13–10 shows the results.

See Also: The online Help for the Edit XPath Expression window, which includes a link to instructions on using the XPath Building Assistant.

See Also: Oracle BPEL Process Manager Order Booking Tutorial for an example of using a `for-each` node.

Note: Elements that display in the XSLT Constructs list and http://www.w3.org/1999/XSL/Transform list of the Component Palette cannot be dragged and dropped into the designer window.
Figure 13–10  Conditional Processing with xsl:if

Conditional Processing with xsl:if

You can copy `PurchaseOrder/HQAccount/AccountNumber` to `Invoice/BilledToAccount/AccountNumber`, if it exists. Otherwise, copy `PurchaseOrder/BranchAccount` to `Invoice/BilledToAccount/AccountNumber` as follows:

1. Select `Invoice/BilledToAccount/AccountNumber` in the target tree and right-click to bring up the context sensitive menu.
2. Select `Add XSL Node`, and then choose and connect `PurchaseOrder/HQAccount/AccountNumber` to `Invoice/BilledToAccount/choose/when` to define the condition.
3. Connect `PurchaseOrder/HQAccount/AccountNumber` to `Invoice/BilledToAccount/choose/when/AccountNumber`.
4. Select XSL Add Node and then choose in the target tree and right-click to bring up the context sensitive menu.
5. Select XSL node and then otherwise from the menu.
6. Connect `PurchaseOrder/BranchAccount/AccountNumber` to `Invoice/BilledToAccount/choose/otherwise/AccountNumber`.

Figure 13–11 shows the results.
Handling Repetition or Arrays

The XSLT Mapper allows repeating elements on the source to be copied to repeating elements on the target. For example, copy PurchaseOrder/Items/HighPriorityItems/Item to Invoice/ShippedItems/Item as follows:

1. Select Invoice/ShippedItems/Item in the target tree and right-click to bring up the context sensitive menu.
2. Select Add XSL Node, and then for-each and connect PurchaseOrder/Items/HighPriorityItems/Item to Invoice/ShippedItems/for-each to define the iteration.
3. Connect PurchaseOrder/Items/HighPriorityItems/Item/ProductName to Invoice/ShippedItems/for-each/Item/ProductName.
4. Connect PurchaseOrder/Items/HighPriorityItems/Item/Quantity to Invoice/ShippedItems/for-each/Item/Quantity.
5. Connect PurchaseOrder/Items/HighPriorityItems/Item/USPrice to Invoice/ShippedItems/for-each/Item/PriceCharged.

Figure 13–12 shows the results.
Automatically Mapping Nodes

Mapping nonleaf nodes starts the auto map feature. The system automatically tries to link all relevant nodes under the selected source and target. Try the auto map feature by mapping `PurchaseOrder/ShipTo/Address` to `Invoice/ShippedTo/Address`. All nodes under `Address` are automatically mapped, as shown in Figure 13–13.

![Auto Mapping](image)

The behavior of the auto map can be tuned by altering the settings in Oracle JDeveloper preferences or by right-clicking the transformation window and selecting `Auto Map Preferences`. This displays the window shown in Figure 13–14.

![Auto Map Preferences](image)

This window enables you to customize your auto mapping as follows:
Using the XSLT Mapper

- Invoke the automatic mapping feature, which attempts to automatically link all relevant nodes under the selected source and target. When disabled, you must individually map relevant nodes.
- Display and review all potential source-to-target mappings detected by the XSLT Mapper, and then confirm to create them.
- Be prompted to customize the auto map preferences before the auto map is invoked.
- Select the Basic or Advanced method for automatically mapping source and target nodes. This enables you to customize how the XSLT mapper attempts to automatically link all relevant nodes under the selected source and target.
- Manage your dictionaries. The XSLT Mapper uses the rules defined in a dictionary when attempting to automatically map source and target elements.

**See Also:** The online Help for the Auto Map Preferences window by clicking the Help button to see a description of the fields

To see potential source mapping candidates for a target node, right-click the target node, select Show Matches, and click OK in the Auto Map Preferences window. The Auto Map window appears, as shown in Figure 13–15.
Using the XSLT Mapper

When the Confirm Auto Map Results check box shown in Figure 13–14 is selected, a confirmation window appears. If matches are found, the potential source-to-target mappings detected by the XSLT Mapper are displayed, as shown in Figure 13–16. The window enables you to filter one or more mappings.

See Also: The online Help for the Auto Map window by clicking the Help button to see a description of the fields.
Viewing Unmapped Target Nodes

You can view a list of target nodes that are currently unmapped to source nodes. Right click in the mapper pane and select **Completion Status**. This window provides statistics at the bottom about the number of unmapped target nodes. This window enables you to identify and correct any unmapped nodes before you test your transformation mapping logic on the Test XSL Map window. Select a target node in the list. The node is highlighted. A check mark indicates that the target node is required to be mapped. If not required, the check box is empty.

**Figure 13–17** provides an example of the Completion Status window.

**See Also:** The online Help for the Auto Map window by clicking the Help button to see a description of the fields.
Generating Dictionaries

A dictionary is an XML file that captures the synonyms for mappings. Right-click the mapper pane and select **Generate Dictionary**. This prompts you for the dictionary name and the directory in which to place the dictionary. The XSLT Mapper uses the rules defined in the dictionary when attempting to automatically map source and target elements. For example, you may want to map a purchase order to a purchase order acknowledgment, then reuse most of the map definitions later:

1. Build all the mapping logic for the purchase order and purchase order acknowledgment.
2. Generate a dictionary for the created map.
3. Create a new map using a different purchase order and purchase order acknowledgment.
4. Load the previously created dictionary by selecting **Preferences > XSL Maps > Auto Map** in the **Tools** main menu of Oracle JDeveloper.
5. Perform an automatic mapping from the purchase order to the purchase order acknowledgment.

Creating Map Parameters and Variables

You can create map parameters and variables. You create map parameters in the source tree and map variables in the target tree.

Note the following issues:

- Parameters are created in the source tree, are global, and can be used anywhere in the mappings.
Variables are created in the target tree, and are either global or local. Where they
are defined in the target tree determines if they are global or local.

- Global variables are defined immediately below the <target> node and
  immediately above the actual target schema (for example, POAcknowledge).
  Right-click on the <target> node to create a global variable.

- Local variables are defined on a specific node below the actual target schema
  (for example, subnode name on schema POAcknowledge). Local variables
  can have the same name as long as they are in different scopes. Local variables
  can only be used in their scopes, while global variables can be used anywhere
  in the mappings.

Creating a Map Parameter

1. Right-click the source tree root and select Add Parameter.
   The Create Parameter window appears.

2. Specify details:

3. Click OK.

Creating a Map Variable

1. Right-click the target tree root and select Add Variable. If you right-click a node
   below the target tree root, select Insert Variable.
   The Create Variable window appears.

2. Specify details:
3. Click OK.

Searching Source and Target Nodes

You can search source and target nodes. For example, you can search in a source node named invoice for all occurrences of the subnode named price.

1. Right-click a source or target node.
2. Enter a keyword for which to search.
3. Specify additional details, as necessary. For example:
   - Select Search Annotations if you want annotations text to also be searched.
   - Specify the scope of the search. You can search the entire source or target tree, search starting from a selected position, or search within a selected subtree.
The first match found is highlighted, and the Find window closes. If no matches are found, a message displays on-screen.

4. Select the F3 key to find the next match in the direction specified. To search in the opposite direction, select the Shift and F3 keys.

Note: You cannot search on functions or text values set with the Set Text option.

Ignoring Elements in the XSLT Document

When the XSLT Mapper encounters any elements in the XSLT document that cannot be found in the source or target schema, it is unable to process them and displays an Invalid Source Node Path error. XSL map generation fails. You can create and import a file that directs the XSLT Mapper to ignore and preserve these specific elements during XSLT parsing by selecting Preferences > XSL Maps in the Tools main menu of Oracle JDeveloper.

For example, preprocessing may create elements named myElement and myOtherElementWithNS that you want the XSLT Mapper to ignore when it creates the graphical representation of the XSLT document. You create and import a file with these elements to ignore that includes the following syntax:

```xml
<elements-to-ignore>
  <element name="myElement"/>
  <element name="myOtherElementWithNS" namespace="NS"/>
</elements-to-ignore>
```

You must restart Oracle JDeveloper after importing the file.
Replacing a Schema in the XSLT Mapper
You can replace the map source schema and map target schema that currently display in the XSLT Mapper. Right-click in either the source or target panel and select Replace Schema. This opens the Select Source and Target Schema window shown in Figure 13–18, which enables you to select the new source or target schema to use.

Figure 13–18 Replacing a Schema

Testing the Map
The XSLT Mapper provides a test utility to test the style sheet or map. The test tool can be invoked by selecting the Test menu item from the mapper, as shown in Figure 13–19.
Testing the Map

Figure 13–19 Invoking the Test Window

Test XSL Map Window

The Test XSL Map window shown in Figure 13–20 enables you to test the transformation mapping logic you designed with the XSLT Mapper. The test settings you specify are stored and do not need to be entered again the next time you test. Test settings must be entered again if you close and reopen Oracle JDeveloper.
1. Choose to allow a sample source XML file to be generated for testing or click Browse to specify a different source XML file in the Source XML File field. When you click OK, the source XML file is validated. If validation passes, transformation occurs, and the target XML file is created. If validation fails, no transformation occurs and a message displays on-screen.

2. Select the Generate Source XML File check box to create a sample XML file based on the map source XSD schema.

3. Select the Show Source XML File check box to display the source XML file for the test. The source XML file displays in an Oracle JDeveloper XML editor. If the map has defined parameters, the Parameters table appears. If you want to specify a value, click Specify Value and make appropriate edits to the Type and Value columns.

4. Enter a file name in the Target XML File field or browse for a file name in which to store the resulting XML document from the transformation.

5. Select the Show Target XML File check box to display the target XML file for the test. The target XML file displays in an Oracle JDeveloper XML editor.

6. If you select to show both the source and target XML, you can customize the layout of your XML editors. Select Enable Auto Layout in the upper right corner and click one of the patterns.

7. Click OK.

The test results appear.

For this example, the source XML and target XML display side-by-side, with the XSL map underneath (the default setting). You can right-click an editor and select Validate XML to validate the source or target XML against the map source or target XSD schema.
Generating Reports

You can generate an HTML report with the following information:

- XSL map file name, source and target schema file names, their root element names, and their root element namespaces
- Target document mappings
- Target fields not mapped (including mandatory fields)
- Sample transformation map execution

To generate a report, right-click the transformation window and select **Generate Report**. The Generate Report window appears in the transformation window, as shown in Figure 13–21. Note that if the map has defined parameters, the **Parameters** table appears.
Figure 13–21  The Generate Report Window

Correcting Memory Errors When Generating Reports
If you attempt to generate a report and receive an out-of-memory error, increase the heap size of the JVM as follows:
1. Open the JDev_Oracle_Home\jdev\bin\jdev.conf file.
2. Go to the following section:
   
   # Set the maximum heap to 512M
   
   AddVMOption -Xmx512M
3. Increase the size of the heap as follows (for example, to 1024)
   
   AddVMOption -Xmx1024M

In addition, you can also uncheck the Open Report option on the Generate Report window before generating the report.

Sample XML Generation
You can customize sample XML generation by specifying the following parameters. Select Preferences > XSL Maps in the Tools main menu of Oracle JDeveloper to display the Preferences window.

- Number of repeating elements
  Specifies how many occurrences of an element are created if the element has the attribute maxOccurs set to a value greater than 1. If the specified value is greater than the value of the maxOccurs attribute for a particular element, the number of occurrences created for that particular element is the maxOccurs value, not the specified number.
Summary

- Generate optional elements
  If selected, any optional element (its attribute `minOccurs` set to a value of 0) is generated the same way as any required element (its attribute `minOccurs` set to a value greater than 0).

- Maximum depth
  To avoid the occurrence of recursion in sample XML generation caused by optional elements, specify a maximum depth in the XML document hierarchy tree beyond which no optional elements are generated.

Summary

This chapter describes features of the XSLT Mapper, such as:
- Creating an XSL map file
- Copying by linking nodes
- Creating functions
- Chaining functions
- Editing XPath expressions
- Adding XSLT constructs such as if, choose, otherwise, and for-each
- Automatically mapping target and source nodes
- Viewing unmapped target nodes
- Generating dictionaries
- Creating map parameters and variables
- Searching source and target nodes
- Ignoring elements in an XSLT document
- Replacing schemas in the XSLT Mapper
- Testing mappings
- Generating mapping reports

This chapter shows these features by providing step-by-step instructions for mapping a sample purchase order schema to an invoice schema.
The notification service in Oracle BPEL Process Manager enables you to send notifications from a BPEL process using a variety of channels. Oracle BPEL Process Manager can deliver these notifications by e-mail, voice message, fax, pager, or short message service (SMS).

This chapter contains the following topics:

- Use Cases for Notification Service
- Overview of Notification Service Concepts
- Configuring the Notification Service in Oracle JDeveloper
- Summary

Use Cases for Notification Service

Various scenarios may require sending e-mail messages or other types of notifications to users as part of the process flow. For example, certain types of exceptions that cannot be handled automatically may require manual intervention. In this case, Oracle JDeveloper uses the notification service to alert users by voice, SMS, fax, pager, or e-mail. In an approval workflow (for example, an expense report approval), you can send notifications to the task assignee when a specific task requires action, or you can notify the task creator by e-mail when the approval is complete. In some cases, contact information (e-mail address or telephone number) is obtained dynamically as part of the process and in other cases the details are looked up from a user directory.

The tutorial `130.SendEmailWithAttachments` demonstrates how to model a notification in Oracle JDeveloper and send an e-mail with an attachment.

See: SOA_Oracle_
Home\bpe\samples\tutorials\130.SendEmailWithAttachments

The OrderBooking tutorial demonstrates how to add an e-mail notification to the POAcknowledge process.

See: Oracle BPEL Process Manager Order Booking Tutorial

Overview of Notification Service Concepts

Terms used for the notification service include:
■ Notification—an asynchronous message sent to a user by a specific channel. The message can be sent as an e-mail message, a voice message, a fax message, a pager message, or an SMS message.

■ Actionable notification—a notification to which the user can respond. For example, workflow sends an e-mail message to a manager to approve or reject a purchase order. The manager approves or rejects the request by replying to the e-mail with appropriate content.

■ Oracle Application Server Wireless—the wireless and voice component of Oracle Application Server. OracleAS Wireless includes a messaging component that handles the sending and receiving of messages to and from devices. When you install OracleAS Wireless, you can specify one of the following notification service options:
  – Connect to an external server to deliver messages, such as e-mail, SMS, fax, voice, or pager.
  – Use Oracle’s hosted service at http://messenger.oracle.com/

Oracle BPEL Process Manager is preconfigured to send notifications using Oracle’s hosted wireless service.

– The notification service supports sending e-mail through the SMTP protocol and receiving e-mail from IMAP- and POP-based e-mail accounts.

Figure 14–1 shows the notification service interfaces and supported service types.

**Figure 14–1 Notification Service Interfaces and Supported Service Types**

![Diagram of notification service interfaces and supported service types]

**Reliable Notification Service**

Oracle BPEL Process Manager provides support for the reliable notification service. The outbound notification service creates a notification message with a unique notification ID and stores the message and unique ID in the dehydration store. It then enqueues this unique ID in the JMS queue and commits the transaction. A message driven-bean (MDB) listening on this queue dequeues the message and sends a notification to the user. If there is any notification failure, the notification service retries three times. If the retries all fail, it marks this notification as errored.
To send an error notification after resolving the problem, you must write a script to update the BPELNotification table status to SEND. For example:

```
UPDATE BPELNotification
SET status = 'RETRY',
    ATTEMPTNUMBER = 0
WHERE ID = <notification id>
```

By default, the notification service retries three times. If you want to add more retries (for example, 5), add the following property in $OA_Oracle_Home/bpel/system/services/config/wf_config.xml and restart Oracle BPEL Server:

```
<property name="oracle.bpel.services.notification.maxattempt" value="5" />
```

The notification thread that is running tries to send the notification every 15 minutes. You can change this interval by adding the following property in wf_config.xml. For example, to retry every 10 minutes:

```
<property name="oracle.bpel.services.notification.publisher_interval" value="10" />
```

---

**Configuring the Notification Service in Oracle JDeveloper**

The diagram window in Oracle JDeveloper includes the notification channels in the Component Palette, as shown in Figure 14–2.

*Figure 14–2  Diagram Window in Oracle JDeveloper—Notification Activity*

To use a notification channel, do the following:

1. Select Process Activities from the Component Palette list.
2. Drag and drop a notification channel from the Component Palette list:
   - Email
   - Fax
   - Pager
Configuring the Notification Service in Oracle JDeveloper

- SMS
- Voice

3. See the following section based on the notification channel you selected.

<table>
<thead>
<tr>
<th>If You Selected...</th>
<th>See...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Email</td>
<td>“The E-mail Notification Channel” on page 14-4 to configure e-mail notification</td>
</tr>
<tr>
<td>Fax</td>
<td>“The Fax Notification Channel” on page 14-8 to configure fax notification</td>
</tr>
<tr>
<td>Pager</td>
<td>“The Pager Notification Channel” on page 14-10 to configure pager notification</td>
</tr>
<tr>
<td>SMS</td>
<td>“The SMS Notification Channel” on page 14-11 to configure SMS notification</td>
</tr>
<tr>
<td>Voice</td>
<td>“The Voice Notification Channel” on page 14-12 to configure voice message notification</td>
</tr>
</tbody>
</table>

The E-mail Notification Channel

When you select Email from the Component Palette, the Edit Email window appears. Figure 14-3 shows the required e-mail notification parameters.

Figure 14–3 Edit Email Window

1. Enter information for each field as described in Table 14–1.

Table 14–1 E-mail Notification Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>From Account</td>
<td>The name of the account used to send this message. The configuration details for this e-mail account name must exist on Oracle BPEL Server.</td>
</tr>
</tbody>
</table>
2. Click OK.

   The BPEL fragment that invokes the notification service to send the e-mail message is created.

See Also: Oracle BPEL Process Manager Administrator's Guide for details about e-mail configuration instructions to perform outside of Oracle JDeveloper

### Setting E-mail Attachments

When you send e-mail attachments, you mark the e-mail as a multipart message and set the number of attachments to send. The number of attachments includes the body plus the attachments. (For example, to send an e-mail message with one file as an attachment, set the number to 2.) When sending attachments, set the content body to have a `Multipart` element that contains as many `BodyPart` elements as the number of attachments. Each `BodyPart` has three elements: `ContentBody`, `MimeType`, and `BodyPartName`. All three elements must be set for each attachment.

To add one attachment to an e-mail message, do the following:

1. Select `Email` as the notification channel from the `Component Palette`.
2. Specify values for `To`, `Subject`, and `Body`.
3. Select `Multipart message` and enter 2 for the number of attachments. (Note that the number of attachments must include the body part.)

   The `Multipart` element with two body parts is generated. The first body part is for the message body and the other is used for the attachment. The BPEL fragment with an `assign` activity with multiple `copy` rules is generated. One of the `copy` rules copies the attachment, as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>To</td>
<td>The e-mail address to which the message is to be delivered. This can be a static e-mail address entered at the time the message is created, or a dynamic address from the identity service, or a dynamic address from the input. The XPath Expression Builder can be used to get the dynamic e-mail address from the input. See Set E-mail Addresses and Telephone Numbers Dynamically on page 14-13.</td>
</tr>
<tr>
<td>CC and Bcc</td>
<td>The e-mail addresses to which the message is copied and blind copied. This can be static or dynamic address as described for the To address.</td>
</tr>
<tr>
<td>Reply To</td>
<td>The e-mail address to use for replies. This can be a static or dynamic address as described for the To address.</td>
</tr>
<tr>
<td>Subject</td>
<td>Subject of the e-mail message. This can be free text or dynamic text. The XPath Expression Builder can be used to set dynamic text based on data from process variables that you specify.</td>
</tr>
<tr>
<td>Body</td>
<td>Message body of the e-mail message. This can be plain text, XML, free text, or dynamic text, as described for the Subject parameter.</td>
</tr>
<tr>
<td>Multipart message with n attachments</td>
<td>Select to specify e-mail attachments. See Setting E-mail Attachments on page 14-5. The number of attachments if Multipart message is selected. The number includes the body. For example, if you have a body and one attachment, specify 2 here.</td>
</tr>
</tbody>
</table>

Table 14–1 (Cont.) E-mail Notification Parameters

Oracle BPEL Process Manager Notification Service 14-5
Configuring the Notification Service in Oracle JDeveloper

<assign name="Assign">
  <copy>
    <from expression="string('Default')"/>
    <to variable="varNotificationReq" part="EmailPayload" query="/EmailPayload/ns1:FromAccountName"/>
  </copy>
  ...
</assign>

<!-- copy statements relate to body and attachment -->

<copy>
  <from>
    <Content xmlns="http://xmlns.oracle.com/ias/pchpel/NotificationService">
      <MimeType xmlns="http://xmlns.oracle.com/ias/pchpel/NotificationService">multipart/mixed</MimeType>
      <ContentBody xmlns="http://xmlns.oracle.com/ias/pchpel/NotificationService">
        <MultiPart xmlns="http://xmlns.oracle.com/ias/pchpel/NotificationService">
          <BodyPart xmlns="http://xmlns.oracle.com/ias/pchpel/NotificationService">
            <MimeType xmlns="http://xmlns.oracle.com/ias/pchpel/NotificationService"/>
            <ContentBody xmlns="http://xmlns.oracle.com/ias/pchpel/NotificationService"/>
            <BodyPartName xmlns="http://xmlns.oracle.com/ias/pchpel/NotificationService"/>
          </BodyPart>
          <BodyPart xmlns="http://xmlns.oracle.com/ias/pchpel/NotificationService">
            <MimeType xmlns="http://xmlns.oracle.com/ias/pchpel/NotificationService"/>
            <ContentBody xmlns="http://xmlns.oracle.com/ias/pchpel/NotificationService"/>
            <BodyPartName xmlns="http://xmlns.oracle.com/ias/pchpel/NotificationService"/>
          </BodyPart>
        </MultiPart>
      </ContentBody>
    </Content>
  </from>
  <to variable="varNotificationReq" part="EmailPayload" query="/EmailPayload/ns1:Content"/>
</copy>

<copy>
  <from expression="string('text/html')"/>
  <to variable="varNotificationReq" part="EmailPayload" query="/EmailPayload/ns1:Content/ns1:ContentBody/ns1:MultiPart/ns1:BodyPart[1]//ns1:MimeType"/>
</copy>

<copy>
  <from expression="string('NotificationAttachment1.html')"/>
  <to variable="varNotificationReq" part="EmailPayload" query="/EmailPayload/ns1:Content/ns1:ContentBody/ns1:MultiPart/ns1:BodyPart[1]/ns1:BodyPartName"/>
</copy>

<copy>
  <from expression="string('This is a test message from John Cooper')"/>
  <to variable="varNotificationReq" part="EmailPayload" query="/EmailPayload/ns1:Content/ns1:ContentBody/ns1:
MultiPart/ns1:BodyPart[1]/ns1:ContentBody/>
</copy>
</copy>
<copy>
   <from expression="string('text/html')"/>
   <to variable="varNotificationReq" part="EmailPayload"
query="/EmailPayload/ns1:Content/ns1:ContentBody/ns1:MultiPart/ns1:BodyPart[2]/
ns1:MimeType/>
</copy>
</copy>
<copy>
   <from expression="string('NotificationAttachment2.html')"/>
   <to variable="varNotificationReq" part="EmailPayload"
query="/EmailPayload/ns1:Content/ns1:ContentBody/ns1:MultiPart/ns1:BodyPart[2]/
nsl:BodyPartName"/>
</copy>
<copy>
   <from expression="string('message2')"/>
   <to variable="varNotificationReq" part="EmailPayload"
query="/EmailPayload/ns1:Content/ns1:ContentBody/ns1:MultiPart/ns1:BodyPart[2]/
nsl:ContentBody"/>
</copy>
</assign>

4. Search for BodyPart[2] and set the required values. The steps to send the
attachment MyImage.gif, for example, are as follows:

a. Search for BodyPart[2] MimeType and change the from expression to
copy 'image/gif' as the MimeType (instead of the autogenerated
'text/html').

b. Search for BodyPart[2] BodyPartName and change the from
expression to copy 'MyImage.gif' (instead of the autogenerated
'NotificationAttachment2.html').

c. Search for BodyPart[2] ContentBody and change the from expression
to copy the content of MyImage.gif (instead of the autogenerated expression
string('message2')).

You can use the readFile XPath function to read the contents of the file:
ora:readFile('<name of the file in the project | HTTP URL | File URL>')

Examples:
ora:readFile('MyImage.gif') will read the file from the bpel project
directory
ora:readFile('file:///c:/MyImage.gif') will read file from c:\ directory
ora:readFile('http://www.oracle.com/MyImage.gif')

The new BPFL copy statement is as follows:
<copy>
   <from expression="string('image/gif')"/>
   <to variable="varNotificationReq" part="EmailPayload"
query="/EmailPayload/ns1:Content/ns1:ContentBody/ns1:MultiPart/ns1:BodyPart[2]/ns1:MimeT
ype"/>
</copy>
</copy>
<copy>
   <from expression="string('MyImage.gif')"/>

Oracle BPEL Process Manager Notification Service 14-7
Configuring the Notification Service in Oracle JDeveloper

Formatting the Body of an E-mail Message as HTML

You can format the body of an e-mail message as HTML rather than as straight text. To do this, apply an XSLT transform to generate the e-mail body. Add in the XSLT tag you want to use. Tools such as XMLSpy can provide assistance in writing and testing the XSLT. The MIME type should be string('text/html;charset=UTF-8').

The e-mail notification assignment should look as follows:

```xml
<to variable="varNotificationReq" part="EmailPayload" query="(EmailPayload/ns1:Content/ns1:ContentBody/ns1:MultiPart/ns1:BodyPart[2]/ns1:BodyPartName)"/>
</copy>
<from expression="ora:readFile('file:///c:/MyImage.gif')"/>
<to variable="varNotificationReq" part="EmailPayload" query="(EmailPayload/ns1:Content/ns1:ContentBody/ns1:MultiPart/ns1:BodyPart[2]/ns1:ContentBody)"/>
</copy>

See: SOA_Oracle_ Home\bpe\samples\tutorials\SendEmailWithAttachments for an example of sending attachments using e-mail

The Fax Notification Channel

When you select Fax from the Component Palette, the Edit Fax window appears. Figure 14–4 shows the required fax notification parameters.
1. Enter information for each field as described in Table 14–2.

Table 14–2 Fax Notification Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fax Number</td>
<td>The fax number to which the message is to be delivered. This can be a static fax number entered at the time the message is created, or a fax number looked up using the identity service, or a dynamic fax number from the payload. The XPath Expression Builder can be used to get the dynamic fax number from the input.</td>
</tr>
<tr>
<td>Cover Page</td>
<td>The cover page name. The cover page details must exist on the server. The cover page can be in PDF, Microsoft Word, HTML, or plain text format (This field is optional.) The XPath Expression Builder can be used to set dynamic text based on data from process variables that you specify.</td>
</tr>
<tr>
<td>Body</td>
<td>Fax message body. This must be plain text or HTML. This can be free text or dynamic text as described for the Cover page parameter.</td>
</tr>
</tbody>
</table>

2. Click OK

The BPEL fragment that invokes the notification service for fax notification is created.

See Also: Oracle BPEL Process Manager Administrator’s Guide for details about fax configuration instructions to perform outside of Oracle JDeveloper
The Pager Notification Channel

When you select Pager from the Component Palette, the Edit Pager window appears. Figure 14–5 shows the required pager notification parameters.

Figure 14–5 Edit Pager Window

1. Enter information for each field as described in Table 14–3.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>From Number</td>
<td>The pager number from which the message is to be sent. This can be a static pager number entered at the time the message is created, or a dynamic pager number from the payload. The XPath Expression Builder can be used to get the dynamic pager number from the input.</td>
</tr>
<tr>
<td>Pager Number</td>
<td>The number of the recipient of this message. This can be a static pager number entered at the time the message is created, or a dynamic pager number from the payload. The XPath Expression Builder can be used to get the dynamic pager number from the input.</td>
</tr>
<tr>
<td>Body</td>
<td>Pager message body. This must be plain text. This can be free text or dynamic text as described for the From Number parameter.</td>
</tr>
</tbody>
</table>

2. Click OK.

The BPEL fragment that invokes the notification service for pager notification is created.
The SMS Notification Channel

When you select SMS from the Component Palette, the Edit SMS window appears. Figure 14–6 shows the required SMS notification parameters.

Figure 14–6 Edit SMS Window

1. Enter information for each field as described in Table 14–4.

Table 14–4 SMS Notification Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>From number</td>
<td>The telephone number from which to send the SMS notification. This can be a static telephone number entered at the time the message is created or a dynamic telephone number from the payload. The XPath Expression Builder can be used to get the dynamic telephone number from the input. See “Setting E-mail Addresses and Telephone Numbers Dynamically” on page 14-13.</td>
</tr>
<tr>
<td>Telephone number</td>
<td>The telephone number to which the message is to be delivered. This can be a) a static telephone number entered at the time the message is created, or b) a telephone number looked up using the identity service, or c) a dynamic telephone number from the payload. The XPath Expression Builder can be used to get the dynamic telephone number from the input.</td>
</tr>
<tr>
<td>Subject</td>
<td>Subject of the SMS message. This can be free text or dynamic text. The XPath Expression Builder can be used to set dynamic text based on data from process variables that you specify.</td>
</tr>
<tr>
<td>Body</td>
<td>SMS message body. This must be plain text. This can be free text or dynamic text as described for the Subject parameter.</td>
</tr>
</tbody>
</table>
2. Click OK.

The BPEL fragment that invokes the notification service for SMS notification is created.

**See Also:** Oracle BPEL Process Manager Administrator’s Guide for details about SMS configuration instructions to perform outside of Oracle JDeveloper

### The Voice Notification Channel

When you select Voice from the Component Palette, the Edit Voice window appears. Figure 14–7 shows the required voice notification parameters.

![Edit Voice Window](image.png)

1. Enter information for each field as described in Table 14–5.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Telephone number</td>
<td>The telephone number to which the message is to be delivered. This can be a) a static telephone number entered at the time the message is created, or b) a telephone number looked up using the identity service, or c) a dynamic telephone number from the payload. The XPath Expression Builder can be used to get the dynamic telephone number from the input.</td>
</tr>
<tr>
<td>Body</td>
<td>Message body. This can be plain text or XML. Also, this can be free text or dynamic text. The XPath Expression Builder can be used to set dynamic text based on data from process variables that you specify.</td>
</tr>
</tbody>
</table>

2. Click OK.
The BPEL fragment that invokes the notification service for voice notification is created.

**See Also:** Oracle BPEL Process Manager Administrator’s Guide for details about voice configuration instructions to perform outside of Oracle JDeveloper

### Setting E-mail Addresses and Telephone Numbers Dynamically

You may need to set e-mail addresses or telephone numbers dynamically based on certain process variables. You can also look up contact information for a specific user using the built-in XPath functions for the identity service.

- To get the e-mail address or telephone number directly from the payload, use the following XPath:

  ```xml
  bpws:getVariableData('<variable name>', '<part>', '<input xpath to get an address>')
  ```

  For example, to get the e-mail address from variable `inputVariable` and part `payload` based on XPath `/client:BPELProcessRequest/client/mail`:

  ```xml
  <%bpws:getVariableData('inputVariable','payload','/client:BPELProcessRequest/client:email')%>
  ```

  You can use the XPath Expression Builder to select the function and enter the XPath expression to get an address from the input variable.

- To get the e-mail address or telephone number dynamically from the payload, use the following XPath:

  ```xml
  ids:getUserProperty(userName, attributeName, realmName)
  ```

  The first argument evaluates to the user ID. The second argument is the property name. The third argument is the realm name. Table 14–6 lists the property names that can be used in this XPath function.

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mail</td>
<td>Look up a user’s e-mail address</td>
</tr>
<tr>
<td>telephoneNumber</td>
<td>Look up a user’s telephone number</td>
</tr>
<tr>
<td>mobile</td>
<td>Look up a user’s mobile telephone number</td>
</tr>
<tr>
<td>homephone</td>
<td>Look up a user’s home telephone number</td>
</tr>
</tbody>
</table>

The following example gets the e-mail address of the user identified by the variable `inputVariable`, part `payload`, and query `/client:BPELProcessRequest/client:userID`:

```xml
ids:getUserProperty(bpws:getVariableData('inputVariable', 'payload', '/client:BPELProcessRequest/client:userID'), 'mail')
```

If `realmName` is not specified, then the default realm name is used. For example, if the default realm name is `jazn.com`, the following XPath expression searches for the user in the `jazn.com` realm:

```xml
ids:getUserProperty('jcooper', 'mail');
```
Configuring the Notification Service in Oracle JDeveloper

The following XPath expression provides the same functionality as the one above. In this case, however, the realm name of jazn.com is explicitly specified:

\[ \text{ids:getUserProperty('jcooper', 'mail', 'jazn.com')} \]

Selecting Notification Recipients by Browsing the User Directory

You can select users or groups to whom you want to send notifications by browsing the user directory (OID, JAZN/XML, LDAP, and so on) that is configured for use by Oracle BPEL Process Manager. Click the first icon (the flashlight) to the right of To (or any recipient field) on any assignee window to start the Identity lookup dialog.

See Also: Chapter 15, "Oracle BPEL Process Manager Workflow Services" for additional details about using the Identity lookup dialog

Starting Business Processes with the E-mail Activation Agent

Activation agents define process agents that initiate a process. You use the e-mail activation agent element \texttt{activationAgents} to start business processes by e-mail. The following steps are required to design a business process to start by e-mail.

1. Create a business process.
2. Add the e-mail activation agent \texttt{activationAgents} element to \texttt{bpel.xml}.
   - See Table 14-7, "E-mail Activation Element and Respective Attributes in \texttt{bpel.xml}" and "The activationAgents Element Structure in \texttt{bpel.xml}" on page 14-14.
3. Include a corresponding account name configuration file in the project.
   - Name the file the same as the name of the \texttt{accountName} attribute of \texttt{activationAgents} in \texttt{bpel.xml}. See "The accountName XML File Structure" on page 14-15.

Table 14-7 describes the \texttt{activationAgents} element and \texttt{activationAgent} attributes of the activation fragment contained in the \texttt{bpel.xml} file.

### Table 14-7 E-mail Activation Element and Respective Attributes in \texttt{bpel.xml}

<table>
<thead>
<tr>
<th>Element/Attribute Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{/activationAgents/activationAgent[className]}</td>
<td>Name of the activation agent class. Use the \texttt{com.collaxa.cube.activation.mail.MailActivationAgent} class as the activation agent.</td>
</tr>
<tr>
<td>\texttt{/activationAgents/activationAgent[heartBeatInterval]}</td>
<td>Polling interval of the messages in seconds</td>
</tr>
<tr>
<td>\texttt{/activationAgents/activationAgent/property name=&quot;accountName&quot;}</td>
<td>Name of the e-mail configuration file. For example, if the account name is \texttt{test_account}, then the \texttt{test_account.xml} file is included in all the e-mail-related information.</td>
</tr>
</tbody>
</table>

The activationAgents Element Structure in \texttt{bpel.xml}

The following code example shows the structure of the \texttt{activationAgents} element contained in \texttt{bpel.xml}.

```xml
<activationAgents>
  <activationAgent
    className="com.collaxa.cube.activation.mail.MailActivationAgent"
    heartBeatInterval="60">
```

14-14 Oracle BPEL Process Manager Developer's Guide
The accountName XML File Structure

The following code example shows the structure of the accountName XML file.

```
<mailAccount xmlns="http://services.oracle.com/bpel/mail/account">
  <userInfo>
    <displayName>\[display name\]</displayName>
    <organization>\[organization name\]</organization>
    <replyTo>\[replyTo email address\]</replyTo>
  </userInfo>
  <outgoingServer>
    <protocol>smtp</protocol>
    <host>\[outgoing smtp server\]</host>
    <authenticationRequired>false</authenticationRequired>
  </outgoingServer>
  <incomingServer>
    <protocol>pop3</protocol>
    <host>\[incoming pop3 server\]</host>
    <email>\[pop user name\]</email>
    <password>\[plain text email password\]</password>
  </incomingServer>
  <!-- IMAP server config -->
  <!--
  <incomingServer>
    <protocol>imap</protocol>
    <host>\[incoming imap server\]</host>
    <email>\[imap user name\]</email>
    <password>\[plain text email password\]</password>
    <folderName>InBox</folderName>
  </incomingServer>
  -->
</mailAccount>
```

XML Validation Failure with the Notification Service

If you set the validateXML property to true (the default is false) on the Manage BPEL Domain window of Oracle BPEL Control, each message exchanged with each of the Web services is validated against its schema. However, notification services fail during XML validity checks at run time. This is because the BPEL variables exchanged with the notification service are not completely initialized in the BPEL process. Part of the initialization happens in the service itself.

Summary

This chapter describes how you can send an e-mail, voice, fax, pager, or short message service (SMS) message from Oracle BPEL Process Manager.
A company’s business processes drive the integration of systems and people that participate in it. The business process and associated systems have a life cycle and certain behavior. The users who participate in the business process have roles and privileges to perform tasks in the business process. Using the workflow services of Oracle BPEL Process Manager, you can blend the integration of systems and services with human workflow into a single end-to-end process flow, while providing visibility and enabling exception handling and optimization at various levels.

This chapter contains the following topics:

- Oracle BPEL Process Manager Workflow Services 10.1.2 and 10.1.3.1.0 Compatibility
- Overview of Workflow Services
- Use Cases for Workflow Services
- Participant Types in Workflow Services
- Overview of the Modeling Process
- Task 1: Creating the Human Task Definition with the Human Task Editor
- Task 2: Associating the Human Task with a BPEL Process
- Task 3: Generating the Task Display Form
- How Changes to a Workflow Appear in Worklist Application
- Notifications from Workflow Services
- End-to-End Workflow Examples
- Workflow Services
- Configuring the Assignment Service
- Workflow Service and Identity Service Related XPath Extension Functions
- NLS Configuration
- Summary
Oracle BPEL Process Manager Workflow Services 10.1.2 and 10.1.3.1.0 Compatibility

See Also:

- Oracle BPEL Process Manager Administrator’s Guide for the organizational hierarchy of the demonstration user community used in examples throughout this chapter
- SOA_Oracle_Home\bpel\system\xml\lib\workflow for workflow service WSDL files
- Oracle BPEL Process Manager Workflow Services API Reference available in SOA_Oracle_Home\bpel\docs\workflow\index.html

Oracle BPEL Process Manager Workflow Services 10.1.2 and 10.1.3.1.0 Compatibility

Workflows that you designed in 10.1.2 with the workflow wizard can be deployed and run in 10.1.3.1.0. However, you must use the old worklist URL to access these tasks:
http://localhost:9700/integration/oldworklistapp/Login

For release 10.1.3.1.0, the workflow wizard has been replaced by a Human Task editor. This editor enables you to specify task settings such as task outcome, payload structure, task participants, assignment and routing policy, expiration and escalation policy, notification settings, and so on.

You cannot use the Human Task editor to edit 10.1.2-based workflows. To use any new 10.1.3.1.0 functionality, the task scope of the workflow must be manually migrated to use the new workflow metadata.

Note also that this is the last release in which you can deploy workflows designed with 10.1.2.

See Also: Appendix E, “Workflow Services Changes Between 10.1.2 and 10.1.3.1” for specific details

Overview of Workflow Services

Workflow services enable you to interleave human interactions with connectivity to systems and services within an end-to-end process flow. As shown in Figure 15–1, workflow services are linked to a BPEL process through a WSDL contract, like any other Web service. The process assigns a task to a user or role and waits for a response. The users act on the task using Oracle BPEL Worklist Application.

Figure 15–1 High-Level View of Workflow Services in Oracle BPEL Process Manager

Terms used in workflow services include:

- Task—work that needs to be done by a user, role, or group

See Also:
Overview of Workflow Services

- Notification—an e-mail, voice, fax, pager, or short message service (SMS) message that is sent when a user is assigned a task or informed that the status of the task has changed
- Worklist—an enumeration of the tasks, or work items, assigned to or of interest to a user
- Human Task editor—A tool that enables you to specify task settings such as task outcome, payload structure, task participants, assignment and routing policy, expiration and escalation policy, notification settings, and so on
- .task file —The metadata task configuration file that stores the task settings specified with the Human Task editor
- routing slip—Contains information about the flow pattern for the workflow, assignees, escalation policy, expiration duration, signature policy, sequence in which the participants interact in the task, and so on.

Features of workflow services include:

- Work queues
  - Standard work queues — high priority tasks, tasks due soon, new tasks, and so on
  - Custom work queues — Users can define new work queues based on specific search criteria
  - Proxy work queues — can grant access to other users to selected work queues. Other users can act on your behalf on those tasks
- Rules Integration
  - User rules — can define custom delegation, auto-approval, or vacation rules
  - Group rules — can define auto-assignment rules for roles or groups; for example, round-robin, least-busy, and so on.
- Task assignment and routing—includes creating tasks from the business process and assigning the tasks to users or roles. Other task assignment and routing features include:
  - Support for task expiration and automatic renewal
  - Support for task delegation, escalation, and reapproval
  - Storage of task history information for auditing, extending workflows to include other workflows, and the ability to archive and purge task details based on specified policies
  - Support for creating custom task escalation rule functions
  - Override and restrict default system actions
  - Specify callback classes on task status
  - JSP-based forms for viewing and updating task details
  - Dynamic assignment functions

See Also:
- “Participant Types in Workflow Services” on page 15-11
- “Dynamic Assignment Functions” on page 15-113
Overview of Workflow Services

■ Built-in reports — Priority reports, productivity reports, cycle time reports, and unattended tasks report

■ Participant types—consists of single approver, group vote, management chain, sequential list of approvers, FYI assignees, and external routing services.

  See Also: “Participant Types in Workflow Services” on page 15-11

■ Identity service—interacts with back-end identity management systems to capture all user information from Java AuthoriZatioN (JAZN) and LDAP. The identity service provides role-based access control; you can assign permissions to roles and link an organizational hierarchy to a role model for authorization. You can also do the following:
  - Assign worklist privileges to users, roles, or groups
  - Maintain user properties such as name, location, phone, fax, and e-mail.
  - Capture organizational hierarchy (reporting structure) and group information
  - Integrate with standard (for example, LDAP-based) directory services for user and role provisioning

  See Also:
  ■ “Identity Service” on page 15-102 for identity service concepts
  ■ Oracle BPEL Process Manager Administrator’s Guide for identity service configuration instructions

■ Notification service
  - Send notifications to specified users on specified task changes
  - Notifications through different delivery channels (e-mail, phone, fax, voice, and SMS)
  - Ability to customize content of notifications for different types of tasks
  - Perform actions on tasks through e-mail

  See Also:
  ■ “Notification Service” on page 15-108 for notification service concepts
  ■ Chapter 14, "Oracle BPEL Process Manager Notification Service"
  ■ Oracle BPEL Process Manager Administrator’s Guide for notification service configuration instructions

■ The Oracle BPEL Worklist Application
  - Out-of-the-box fully customizable worklist
  - Support for various user profiles – end user, supervisor, task owner, group owner, administrator
  - View tasks based on user or role ability to perform authorized actions on tasks in the worklist
  - Ability to filter tasks in worklist view based on various criteria
  - Ability to acquire and check out shared tasks

See Also:
■ Oracle BPEL Process Manager Developer’s Guide
Overview of Workflow Services

- Support for custom work queues
- Define custom vacation rules and delegation rules
- Provide access to selected worklist views to other users (proxy support)
- Complete workflow history and audit trail
- Out-of-the-box productivity reports

See Also: Chapter 16, "Worklist Application"

Workflow Functionality: A Procurement Process Example

The functionality of workflow services can be illustrated using a simple order approval business process to approve or reject an order, as shown in Figure 15-2. The process requested items. Approval and rejection is a two-step process involving an initial approver and the manager of the initial approver. The order is first assigned to the Supervisor role. Once a user belonging to the Supervisor role approves the order, it is sent to this user’s manager for final approval.

Figure 15–2  BPEL Workflow

Oracle BPEL Process Manager Workflow Services 15-5
Overview of Workflow Services

Workflow Services Components

Figure 15–3 shows the following workflow services components:

- Task Service
  The task service provides task state management and persistence of tasks. In addition to these services, the task service exposes operations to update a task, complete a task, escalate and reassign tasks, and so on. The task service is used by the Oracle BPEL Worklist Application to retrieve tasks assigned to users. This service also determines if notifications are to be sent to users and groups when the state of the task changes. The task service consists of the following services.
  - Task Routing Service
    The task routing service offers services to route, escalate, and reassign the task. The service makes these decisions by interpreting a declarative specification in the form of the routing slip.
  - Task Query Service
    The task query service queries tasks for a user based on a variety of search criterion such as keyword, category, status, business process, attribute values, history information of a task, and so on.
  - Task Metadata Service
    The task metadata service exposes operations to retrieve metadata information related to a task.

- Identity Service
  The identity service is a thin Web service layer on top of the Oracle Application Server 10g security infrastructure or any custom user repository. It enables authentication and authorization of users and the lookup of user properties, roles, group memberships, and privileges.

- Notification Service
  The notification service delivers notifications with the specified content to the specified user to any of the following channels: e-mail, telephone voice message, pager, fax, and SMS. See "Notifications from Workflow Services" on page 15-80 for more information.

- User Metadata Service
  The user metadata service manages metadata related to workflow users, such as user work queues, preferences, vacation, and delegation rules.

- Runtime config service
  The runtime config service provides methods for managing metadata used in the task service run time environment. It principally supports management of task payload flex field mappings.

See Also: Oracle BPEL Process Manager Order Booking Tutorial for instructions on designing an order approval business process to approve or reject an order.
Figure 15–3 Workflow Services Components

Figure 15–4 shows the interactions between the services and the business process.
Figure 15–4  Workflow Services and Business Process Interactions

Use Cases for Workflow Services

Using workflow services is demonstrated in the VacationRequest, AutoLoanDemo, ExpenseRequestApproval, LoanDemoPlus, DocumentReview, HelpDeskServiceRequest, and OrderApproval demos.

See Also:  Oracle BPEL Process Manager Administrator’s Guide for identity service details

See Also:
- “End-to-End Workflow Examples” on page 15-86
- SOA_Oracle_Home\bpel\samples\demos
The following sections describe multiple use cases for workflow services.

**Assigning a Task to a User or Role**

A vacation request process may start with getting the vacation details from a user and then routing the request to their manager for approval. User details and the organizational hierarchy can be looked up from a user directory or store. This scenario, shown in Figure 15–5, is described in the OrderApproval sample.

*Figure 15–5 Assigning Tasks to a User or Role from a Directory*

**Using the Various Participant Types**

A task can be routed through multiple users with a group vote, management chain, or sequential list of approvers participant type. For example, consider a loan request that is part of the loan approval flow. The loan request may first be assigned to a loan agent role. After a specific loan agent acquires and accepts the loan, the loan may be routed further through multiple levels of management if the loan amount is greater than $100,000. This scenario, shown in Figure 15–6, is described in the LoanDemoPlus sample.

*Figure 15–6 Flow Patterns and Routing Policies*

See "Participant Types in Workflow Services" on page 15-11 for the various flow types supported by workflow services. You can use these types as building blocks to create complex workflows.
Escalation, Expiration, and Delegation

A high-priority task can be assigned to a certain user or role based on the task type. However, if the user does not act on it in a certain time, the task may expire and in turn be escalated to the manager for further action. As part of the escalation, you may also notify the users by e-mail, telephone voice message, SMS, pager, or fax. Similarly, a manager may delegate tasks from one reportee to another to balance the load between various task assignees. All tasks defined in BPEL have an associated expiration date. Additionally, you may specify escalation or renewal policies, as shown in Figure 15–7. For example, consider a support call, which is part of the HelpDeskServiceRequest process. A high-priority task may be assigned to a certain user and if the user does not respond in two days, then the task is routed to the manager for further action.

Automatic Assignment and Delegation

A user may decide to have another user perform tasks on their behalf. Tasks can be explicitly delegated from the Oracle BPEL Worklist Application or can be automatically delegated. For example, a manager sets up a vacation rule saying that all their high priority tasks are automatically routed to one of their reports while the manager is on vacation. In some cases, tasks can be routed to different individuals based on the content of the task. Another example of automatic routing is to allocate tasks among multiple individuals belonging to a group. For example, a help desk supervisor decides to allocate all tasks for the western region based on a round robin basis or assign tasks to the individual with the lowest number of outstanding tasks (the least busy).

Work Queues and Proxy Support

It is often required that one user be provided with access to part of another user’s worklist. For example, an executive decides to provide access to expense approvals within a certain limit to their secretary. Work queues allow you to create a custom view to group a subset of tasks in the worklist (say high priority tasks, tasks due in 24 hours, expense approval tasks, and so on). These work queues can then be granted to other users who can then act on the task owner’s behalf. For example, in the scenario described above, the executive can create a delegated expense approvals work queue for expenses below $5000.

The Oracle BPEL Worklist Application

Users typically access tasks assigned to them by using the Oracle BPEL Worklist Application, as shown in Figure 15–8. A worklist consists of tasks assigned to the user.
Participant Types in Workflow Services

Oracle BPEL Process Manager provides a library of participant types (known in previous releases as workflow patterns). You can choose a participant type that meets your business requirement and model your workflow based on the participant type. Oracle BPEL Process Manager supports the following participant types:

- **Single Approver** — used for a single user to act on a task. If the task is assigned to a role or group with multiple users, one of the members must claim the task and act on it. Based on the user's action, you define what the business process does.
- **Group Vote** — used when multiple users, working in parallel, must take action simultaneously, such as in a hiring situation when multiple users vote to hire or reject an applicant. You specify the voting percentage that is needed for the outcome to take effect, such as a majority vote or a unanimous vote.
- **Management Chain** — used to route tasks for approval to multiple users in a management chain hierarchy. You specify the task participants as a management chain list or a list of users.
- **Sequential list of approvers** (extension of a sequential workflow) — used to create a list of sequential participants for a workflow. This type is similar to the management chain participant type, except that with that type, the users are part of an organization hierarchy. For the sequential list of approvers participant type, they can be any list of users or groups.
- **FYI assignee** — used when a task is sent to a user, but the business process does not wait for a user response; it just continues. FYI assignees cannot directly impact the outcome of a task, but in some cases can provide comments or add attachments.
Overview of the Modeling Process

- External Routing Service — used to configure an external routing service that dynamically determines the participants in the workflow. If this participant type is specified, all other participant types are ignored. It is assumed that the external routing service provides a list of participant types (single approver, list of approvers, group vote, and so on) at runtime to determine the routing of the task.

Continuing Workflows from Other Workflows

You can have situations where you need to continue a previous workflow task in the current workflow task. Oracle BPEL Process Manager enables you to include the task history, comments, and attachments from the previous task. This provides you with a complete end-to-end audit trail.

See Also: "Including the Task History of Other Human Tasks" on page 15-59

Overview of the Modeling Process

The modeling process consists of creating a human task, associating it with a BPEL process, and generating the format for displaying the human task during runtime in the Oracle BPEL Worklist Application. This section provides a brief overview of these modeling tasks and provides references to specific modeling instructions.

- Create a Human Task Definition with the Human Task Editor
- Associate the Human Task Definition with a BPEL Process
- Generate the Task Display Form

Create a Human Task Definition with the Human Task Editor

The Human Task editor enables you to define the metadata for the task. This editor enables you to specify human task settings, such as task outcome, payload structure, task participants, assignment and routing policy, expiration and escalation policy, notification settings, and so on. This information is saved to a metadata task configuration file with a .task extension.

See Also: "Task 1: Creating the Human Task Definition with the Human Task Editor" on page 15-13 for specific instructions

Associate the Human Task Definition with a BPEL Process

You associate the .task file that consists of the human task settings with a BPEL process. Association is made with a human task activity that you drag and drop into your BPEL process for configuring. You also define the task definition, task initiator, task priority, and map the task parameter that carries the input data to a BPEL variable. You can also define advanced features, such as the scope and global task variables names (instead of accepting the default names), task owner, identification key, BPEL callback customizations, and whether to extend the human task to include other workflow tasks.

When association is complete, a Task Service partner link is created. The Task Service exposes the operations required to act on the task.

See Also: "Task 2: Associating the Human Task with a BPEL Process" on page 15-53 for specific instructions
Generate the Task Display Form

You generate the layout of the task display form used for displaying the task header, body (task payload), and footer details at run time in Oracle BPEL Worklist Application. The task display form defines the display mechanism for the task payload (data in the task) in the Oracle BPEL Worklist Application. Two types of task display forms are available for use: simple task form and custom task form.

See Also: “Task 3: Generating the Task Display Form” on page 15-66 for specific instructions

Task 1: Creating the Human Task Definition with the Human Task Editor

The Human Task editor enables you to define the metadata for the task. This editor enables you to specify human task settings, such as task outcome, payload structure, task participants, assignment and routing policy, expiration and escalation policy, notification settings, and so on.

When human task creation is complete, the following folder and file are created:

- A folder with the human task name you specify in the Human Task Name field in "Accessing the Human Task Editor" on page 15-13 is created under the Integration Content folder of your BPEL process in the Application Navigator
- The human task settings specified in the Human Task editor are saved to a metadata task configuration file with a .task extension. This file is stored in the human task name folder. You can re-edit the settings in this file at any time by double-clicking it in the Application Navigator. This reopens the .task file in the Human Task editor.

This section contains the following topics:

- Accessing the Human Task Editor
- Reviewing the Sections of the Human Task Editor
- Specifying the Task Title, Priority, Outcome, and Owner
- Specifying the Task Payload Data Structure
- Assigning Task Participants
- Escalating, Renewing, or Ending the Task
- Specifying Participant Notification Preferences
- Specifying Advanced Settings
- Exiting the Human Task Editor and Saving Your Changes

Accessing the Human Task Editor

When you are ready to begin creation of a human task, the Human Task editor can be accessed in several ways in Oracle JDeveloper:

- From the Application Navigator
- From the Component Palette

From the Application Navigator

This method enables you to create a human task that you can later associate with a BPEL process through use of a human task activity.
Task 1: Creating the Human Task Definition with the Human Task Editor

1. Right-click your BPEL process in the Application Navigator and select Create Human Task Definition.
   The Add a Human Task window appears.
2. Enter a name in the Human Task Name field.
   The name you enter is added to the directory path in the Location field.
   $OA_Oracle_Home/jdev/mywork/my_application/my_process/bpel/
   Human_task_directory/Human_task_name.task
3. Click OK.
   The Human Task editor appears.
4. Go to section “Reviewing the Sections of the Human Task Editor” on page 15-14.

From the Component Palette
This method enables you to create a human task activity with which you immediately associate a BPEL process through use of a human task activity.

   1. Select Process Activities from the Component Palette.
   2. Drag and drop a Human Task activity into your BPEL process.
      The Add a Human Task window appears.
   3. Click the second icon to the right of the Task Definition field.
      Enter a name in the Human Task Name field.
      The name you enter is added to the directory path in the Location field.
      $OA_Oracle_Home/jdev/mywork/my_application/my_process/bpel/
      Human_task_directory/Human_task_name.task
   5. Click OK.
      The Human Task editor appears.
   6. Go to section “Reviewing the Sections of the Human Task Editor” on page 15-14.

Reviewing the Sections of the Human Task Editor
The Human Task editor consists of the following main sections shown in Figure 15–9.
These sections enable you to create a human task.
Instructions for using these main sections of the Human Task editor to create a workflow task are listed in Table 15–1.

<table>
<thead>
<tr>
<th>For This Main Section...</th>
<th>See...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task Configuration</td>
<td>&quot;Specifying the Task Title, Priority, Outcome, and Owner&quot; on page 15-15</td>
</tr>
<tr>
<td>Parameters</td>
<td>&quot;Specifying the Task Payload Data Structure&quot; on page 15-21</td>
</tr>
<tr>
<td>Assignment and Routing Policy</td>
<td>&quot;Assigning Task Participants&quot; on page 15-22</td>
</tr>
<tr>
<td>Expiration and Escalation Policy</td>
<td>&quot;Escalating, Renewing, or Ending the Task&quot; on page 15-39</td>
</tr>
<tr>
<td>Notification Settings</td>
<td>&quot;Specifying Participant Notification Preferences&quot; on page 15-43</td>
</tr>
<tr>
<td>Advanced Settings</td>
<td>&quot;Specifying Advanced Settings&quot; on page 15-47</td>
</tr>
</tbody>
</table>

Specifying the Task Title, Priority, Outcome, and Owner

Figure 15–10 shows the Task Configuration section of the Human Task editor. This section enables you to specify details such as the task title, task priority, task outcomes, and task owner.
Instructions for configuring the following subsections of the Task Configuration section are listed in Table 15–2:

Table 15–2 Human Task Editor — Task Configuration Section

<table>
<thead>
<tr>
<th>For This Subsection...</th>
<th>See...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title</td>
<td>“Specifying a Task Title and Priority” on page 15-16</td>
</tr>
<tr>
<td>Priority</td>
<td></td>
</tr>
<tr>
<td>Outcomes</td>
<td>“Specifying a Task Outcome” on page 15-16</td>
</tr>
<tr>
<td>Owner</td>
<td>“Specifying a Task Owner” on page 15-18</td>
</tr>
</tbody>
</table>

### Specifying a Task Title and Priority

1. Enter the following details.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title</td>
<td>Enter an optional task title. The task title displays in the Oracle BPEL Worklist Application. If you enter a title in the Task Title field of the General tab of the Human Task window described in “Specifying the Task Title” on page 15-55, the title you enter here is overridden.</td>
</tr>
<tr>
<td>Priority</td>
<td>Specify the priority of the tasks. Priority can be 1 through 5, with 1 being the highest. By default, the priority of a task is 3. The priority can be used to sort tasks in the Oracle BPEL Worklist Application. This priority value is overridden by any priority value you select in the General tab of the Add a Human Task window. See Also: “Specifying the Task Initiator and Task Priority” on page 15-55 for instructions on specifying a priority value in the Add a Human Task window</td>
</tr>
</tbody>
</table>

### Specifying a Task Outcome

Task outcomes capture the possible outcomes of a task. The Oracle BPEL Worklist Application displays the outcomes you specify here as the possible actions to perform during run time. You can specify the following types of task outcomes:

- Select a seeded outcome
- Enter a custom outcome

The task outcomes can also have run time display values that are different from the actual outcome value specified here. This permits outcomes to be displayed in a different language in the Oracle BPEL Worklist Application. See “Specifying Multilingual Settings” on page 15-48 for more information about internationalization.

1. Click the flashlight icon to the right of the Outcomes field.

   The Outcomes window displays the possible outcomes for tasks. APPROVE and REJECT are selected by default.
2. Select additional task outcomes or deselect the default outcomes.
3. Enter any custom outcomes separated by commas in the Custom Outcomes field.
4. Click OK to return to the Human Task editor.

Your selections display in the Outcomes field.

The seeded and custom outcomes selected here display for selection in the Majority Voted Outcome section of the group vote participant type.

See Also: "Specifying Group Voting Details" on page 15-29

Displaying Custom Outcomes in a Human Task Activity The method by which you create a human task definition determines whether custom outcomes initially display in a switch activity. If you perform the following tasks:
1. Drag and drop a human task activity into the design window.
2. Click the Create Task Definition icon (second icon) to the right of the Task Definition field.
3. Create a human task definition with custom outcomes.
4. Expand the human task activity.

Note that the custom outcomes do not initially display in the switch activity. As a workaround, perform the following steps:
1. Click the human task activity to display the Human Task window.
2. Click OK.
3. Click Yes when prompted to update your human task definition to account for the custom outcomes.
4. Click Source.
5. Click Diagram.
6. Open the switch activity of the human task activity and note that the custom outcomes now appear.

Or, always create human task definition files as follows:
Task 1: Creating the Human Task Definition with the Human Task Editor

1. Right-click the BPEL process in the Application Navigator.
2. Select Create Human Task Definition.
3. Design a human task definition.
4. Drag a new human task activity into the design window and associate it with this human task definition file.
5. Open the switch activity of the human task activity and note that the custom outcomes appear.

Specifying a Task Owner

The task owner can view the tasks belonging to business processes they own and perform operations on behalf of any of the assigned task participant types. Additionally, the owner can also reassign, withdraw, or escalate tasks. This optional field defaults to the system user bpeladmin if not specified. The task owner can also be specified in the Advanced tab of the Human Task window described in “Specifying a Task Owner” on page 15-58. The task owner specified in the Advanced tab overrides any task owner you enter here.

1. Select a method for specifying the task owner:
   - Specifying a Task Owner By Browsing the User Directory
   - Specifying a Task Owner Dynamically

Specifying a Task Owner By Browsing the User Directory

Task owners can be selected by browsing the user directory (Oracle Internet Directory (OID), JAZN/XML, LDAP, and so on) that is configured for use with Oracle BPEL Process Manager.

1. Click the first icon to the right of the Owner field to display the Identity lookup dialog.
2. Search for the owner by entering a search string such as jcooper, j*, *, and so on. Clicking Lookup fetches all the users that match the search criteria.
One or more users or groups can be highlighted and selected by clicking **Select**.

3. View the hierarchy of a user by highlighting the user and clicking **Hierarchy**. Similarly, clicking **Reportees** displays the reportees of a selected user or group.
4. View the details of a user or group by highlighting the user or group and clicking Detail.

5. Click OK to return to the Identity lookup dialog.

6. Click Select to add the user to the Selected user section.

7. Click OK to return to the Human Task editor.

Your selection displays in the Owner field.

Specifying a Task Owner Dynamically

Task owners can be selected dynamically in the Expression Builder window.

1. Click the second icon to the right of the Owner field to display the Expression Builder window.
Task 1: Creating the Human Task Definition with the Human Task Editor

2. Browse the available variable schemas and functions to create a task owner.
3. Click OK to return to the Human Task editor. You selection displays in the Owner field.

See Also:
- Click Help for instructions on using the Expression Builder window and XPath Building Assistant
- “Workflow Service and Identity Service Related XPath Extension Functions” on page 15-122 and Appendix D, “XPath Extension Functions” for information about workflow service dynamic assignment functions and identity service functions

Specifying the Task Payload Data Structure

Figure 15–11 shows the Parameters section of the Human Task editor. This section enables you to define the structure (message attributes) of the task payload (the data in the task). Task payload data consists of one or more elements or types. Based on your selections, an XML schema definition is created for the task payload.

Figure 15–11  Human Task Editor — Parameters Section

1. Click the + sign to display the Add Task Parameter window.

2. Enter the following details:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter Type</td>
<td>Select Type or Element and click the flashlight icon to display the Type Chooser window for selecting the task parameter.</td>
</tr>
<tr>
<td>Name</td>
<td>Accept the default name or enter a custom name. This field only displays if Type is the selected parameter type.</td>
</tr>
</tbody>
</table>
Task 1: Creating the Human Task Definition with the Human Task Editor

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modifiable via worklist</td>
<td>Select this check box to enable users to edit task payload data in the footer of the Oracle BPEL Worklist Application. For example, the approver in the application may need to add approver comments.</td>
</tr>
</tbody>
</table>

3. Click OK to return to the Human Task editor. Your selection displays in the Parameters section.
4. If you want to edit your selection, highlight it and click the first icon in the upper right part of the Parameters section.

Assigning Task Participants

Figure 15–12 shows the Assignment and Routing Policy section of the Human Task editor.

This section enables you to select a participant type that meets your business requirement. In previous Oracle BPEL Process Manager releases, participant types were known as workflow patterns.

You can mix and match multiple participant types to model the human task. This enables you to extend the functionality of a previously configured human task to model more complex workflows.

Each of the participant types has an associated editor that you use for configuration tasks. The sequence in which the assignees are added indicates the execution sequence.

**Figure 15–12  Human Task Editor — Assignment and Routing Policy Section**

1. Click the + sign to display the Add Participant Type window. This enables you to select a specific participant type.
2. Select a participant type from the Type list.
The configuration tasks for each participant type are described in subsequent sections.

3. See the following section based on your selection:

<table>
<thead>
<tr>
<th>For This Subsection...</th>
<th>See...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add Participant Type</td>
<td></td>
</tr>
<tr>
<td>■ Single approver</td>
<td>“Configuring the Single Approver Participant Type” on page 15-24</td>
</tr>
<tr>
<td>■ Group vote</td>
<td>“Configuring the Group Vote Participant Type” on page 15-27</td>
</tr>
<tr>
<td>■ Management chain</td>
<td>“Configuring the Management Chain Participant Type” on page 15-30</td>
</tr>
<tr>
<td>■ Sequential list of approvers</td>
<td>“Configuring the Sequential List of Approvers Participant Type” on page 15-32</td>
</tr>
<tr>
<td>■ FYI assignee</td>
<td>“Configuring the FYI Assignee Participant Type” on page 15-35</td>
</tr>
<tr>
<td>■ External routing service</td>
<td>“Configuring the External Routing Service Participant Type” on page 15-36</td>
</tr>
</tbody>
</table>

4. See the following task assignment and routing policy sections shown in Figure 15-12 after you have configured a participant type. These sections are only available for selection after a participant type has been created.

<table>
<thead>
<tr>
<th>For This Subsection...</th>
<th>See...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allow all participants to invite other participants</td>
<td>“Allowing All Participants to Invite Other Participants” on page 15-37</td>
</tr>
</tbody>
</table>

| Enable abrupt completion condition | “Abruptly Completing a Condition” on page 15-38 |

### Specifying Task Approvers

Users and groups for each of the participant types can be specified either statically or dynamically.

When the users and groups are specified statically (or by browsing the identity service), the values can be either of the following:

- A single user or group (for example, jstein), which in the case of a single approver, is captured as follows:

  ```xml
  <participant name="Assignee1">
    <resource isGroup="false" type="STATIC">jstein</resource>
  </participant>
  ```

- A delimited string of users or groups (for example, jstein,wfaulk,cdickens), which in the case of a single approver, is captured as follows:

  ```xml
  <participant name="Assignee1">
    <resource isGroup="false" type="STATIC">jstein,wfaulk,cdickens</resource>
  </participant>
  ```
Task 1: Creating the Human Task Definition with the Human Task Editor

You may have a business requirement to create a dynamic list of task approvers specified in a payload variable. This XPath expression can resolve to zero or more XML nodes. Each node value can be either of the following:

- A single user or group
- A delimited string of users or groups. For example, the following task shows that the payload message attribute is of type `xsd:string` and its value is a comma-delimited string of approvers. This node can be used to specify the participants.

```xml
<participant name="Assignee1">
  <resource isGroup="false" type="STATIC">jstein, wfaulk, cdickens</resource>
</participant>
```

```xml
<task>
  ...
  <payload>
    <approvers>jstein, wfaulk, cdickens</approvers>
  </payload>
</task>
```

The default delimiter for the assignee delimited string is a comma (,). This delimiter can be changed using the `assigneeDelimiter` XML element in the `wf-config.xml` file. This delimiter applies to all workflows in the system.

Specifying participants in this manner is applicable to all participant types, although they are interpreted differently for each type. For example:

- In a single user participant type, the task is assigned to everyone evaluated.
- In a sequential list of approvers participant type, the task is sequentially assigned to users and groups evaluated in the list.
- In a group vote participant type, a task is created for each user and group evaluated in the list.

This interpretation of resource XPath expressions provides `orcl:create-nodeset-from-delimited-string`-equivalent functionality to enable you to specify a dynamic list of one or more task approvers (resource element members) from the payload variable.

Configuring the Single Approver Participant Type

Figure 15–13 displays the Single Approver window.

This participant type requires a single user to act on a task. If the task is assigned to a role or group with multiple users, one of the members must claim the task and act on it. Based on the user's action, you define what the business process does.

For example, a vacation request is assigned to a manager. The manager must act on the request task three days before the vacation starts. If the manager formally approves or rejects the request, the employee is notified with the decision. If the manager does not act on the task, the request is treated as rejected. Notification actions similar to the formal rejection are taken.
Figure 15–13  Add Participant Type — Single Approver

1. Enter a recognizable label for this participant in the **Label** field. This label must be unique within this workflow (for example, Approval Manager, Primary Reviewers, and so on).

Instructions for configuring the following subsections of the Add Participant Type - Single Approver window are listed in **Table 15–3**:

**Table 15–3 Add Participant Type — Single Approver**

<table>
<thead>
<tr>
<th>For This Subsection...</th>
<th>See...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requires action from one of the participants below</td>
<td>“Assigning Participants to the Single Approver Task” on page 15-25</td>
</tr>
<tr>
<td>Specify skip rule</td>
<td>“Bypassing a Task Participant” on page 15-26</td>
</tr>
<tr>
<td>Limit allocated duration to</td>
<td>“Specifying a Time Limit for Acting on a Task” on page 15-26</td>
</tr>
<tr>
<td>Allow this participant to invite other participants</td>
<td>“Inviting Additional Participants to a Task” on page 15-26</td>
</tr>
</tbody>
</table>

**Assigning Participants to the Single Approver Task**

1. Select a method for assigning a user or group to participate in performing actions on this task.

- **By name**
  
Enter a user or group name or click the first icon (**flashlight**) to the right of the field to display a window for selecting a user or group configured through the identity service. The identity service enables user authorization and the
lookup of user properties, roles, group memberships, and privileges. User information is obtained from Java AuthoriZatioN (JAZN) or an LDAP server such as Oracle Internet Directory. You can use wild cards (*) to search for IDs.

- **By expression**

  Dynamically assign this task to a user (for example, `jcooper`) or group (for example, `administrators`) by clicking the icon to the right of the field to display the Expression Builder window. Users who are members of a group are assigned this task. For a user to act on a task assigned to a group, they must first claim the task in the Oracle BPEL Worklist Application during run time.

  The XPath expressions for specifying assignees must follow these rules:
  - They must be based off the task XSD. This includes the payload as defined in the payload section. For example, `/task:task/task:payload/order:orderAssignee` is an example of an XPath expression based of the task XSD.
  - The XPath expressions cannot contain BPEL-specific XPath functions such as `bpws:getVariableData()`, and so on because these XPath expressions are not evaluated from the context of a BPEL instance.
  - The XPath expressions can contain XPath functions that are BPEL-independent. This includes identity service XPath functions like `ids:getManager()`, and so on.

**Bypassing a Task Participant**

1. Select the **Specify skip rule** check box if you want the user or group to be bypassed if a specific condition is satisfied. This action displays an icon for accessing the Expression Builder window for building a condition. For example, if a user submits a business trip expense report that is below a specific amount, no approval is required by their manager.

   The expression to bypass a task participant must evaluate to a Boolean value. For example, `/task:task/task:payload/order:orderAmount < 1000` is a valid XPath expression for skipping a participant.

**Specifying a Time Limit for Acting on a Task**

1. Click the + sign to expand the **Advanced** section shown in Figure 15–13.

2. Select **Limit allocated duration to**.

3. Specify the amount of time a user or group receives to act on a task. If the user or group does not act in the time specified, the global escalation and renewal policies that you set in the **Expiration and Escalation Policy** section (known as the routing slip level) of the Human Task editor are applied. For example, if the global policy is set to escalate the task and this participant does not act in the duration provided, the task is escalated to the manager or another user, as appropriate.

   **See Also:** "Escalating, Renewing, or Ending the Task" on page 15-39 for instructions on setting the global escalation and renewal policies in the **Expiration and Escalation Policy** section of the Human Task editor.

**Inviting Additional Participants to a Task**

1. Click the + sign to expand the **Advanced** section (if not already expanded).
2. Select the **Allow this participant to invite other participants** check box if you want this task assignee to invite other participants into the workflow before routing it to the next assignee in this workflow. For example, assume the approval workflow goes from James Cooper to John Steinbeck. If this option is checked, James Cooper can decide to first route it to Irving Stone before it goes to John Steinbeck.

### Configuring the Group Vote Participant Type

**Figure 15–14** displays the Group Vote window.

This participant type is used when multiple users, working in parallel, must take action simultaneously, such as in a hiring situation when multiple users vote to hire or reject an applicant. You specify the voting percentage that is needed for the outcome to take effect, such as a majority vote or a unanimous vote.

For example, a business process collects the feedback from all interviewers in the hiring process, consolidates it, and assigns a hire or reject request to each of the interviewers. At the end, the candidate is hired if the majority of interviewers vote for hiring instead of rejecting.

**Figure 15–14 Add Participant Type — Group Vote**

1. Enter a recognizable label for this participant in the **Label** field. This label must be unique within this workflow (for example, Approval Manager, Primary Reviewers, and so on).
Instructions for configuring the following subsections of the Add Participant Type - Group Vote window are listed in Table 15–4:

<table>
<thead>
<tr>
<th>For This Subsection</th>
<th>See...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required consensus between the participants below: 50</td>
<td>“Assigning Participants to the Group Vote Task” on page 15-28</td>
</tr>
<tr>
<td>Specify skip rule</td>
<td>“Bypassing a Task Participant” on page 15-28</td>
</tr>
<tr>
<td>Share attachments and comments</td>
<td>“Sharing Attachments and Comments with Task Participants” on page 15-29</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Default Outcome</th>
<th>Consensus Percentage</th>
<th>Immediately trigger voted outcome when minimum percentage is met</th>
<th>Wait until all votes are in before triggering outcome</th>
<th>Limit allocated duration to</th>
<th>See...</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>“Specifying Group Voting Details” on page 15-29</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>“Specifying a Time Limit for Acting on a Task” on page 15-29</td>
</tr>
</tbody>
</table>

Assigning Participants to the Group Vote Task

1. Select a method for assigning a user or group to participate in this task. The assigned participants must establish a consensus on when a task is considered complete.

   - **By name**
     
     Enter a user or group name or click the first icon (flashlight) to the right of the field to display a window for selecting a user or group configured through the identity service. The identity service enables user authorization and the lookup of user properties, roles, group memberships, and privileges. User information is obtained from Java AuthoriZatioN (JAZN) or an LDAP server such as Oracle Internet Directory. You can use wildcards (*) to search for IDs.

   - **By expression**
     
     Dynamically assign this task to a user (for example, jcooper) or group (for example, administrators) by clicking the icon to the right of the field to display the Expression Builder window. Users who are members of a group are assigned this task. For a user to act on a task assigned to a group, they must first claim the task in the Oracle BPEL Worklist Application during run time.

   **See Also:** “Assigning Participants to the Single Approver Task” on page 15-25 for rules to follow when specifying assignees with XPath expressions

Bypassing a Task Participant

1. Select the Specify skip rule check box if you want the user or group to be bypassed if a specific condition is satisfied. This action displays an icon for accessing the Expression Builder window for building a condition. For example, if a user submits a business trip expense report that is below a specific amount, no approval is required by their manager. The expression must evaluate to a Boolean value.
Task 1: Creating the Human Task Definition with the Human Task Editor

Sharing Attachments and Comments with Task Participants

1. Select the Share attachments and comments check box if you want all group voters or workflow participants to share comments and attachments for this task. This information typically displays in the footer region of the Oracle BPEL Worklist Application.

Specifying Group Voting Details

1. Specify a method for selecting the outcome for the final task. If you select By Expression from the lists below, you can dynamically specify the details by clicking the icon to the right of the field to display the Expression Builder window.

   ■ Default Outcome
   Select the default outcome for this task to take effect if the consensus percentage value is not satisfied. This happens if there is a tie or if all participants do not respond before the task expires. Seeded and custom outcomes that you entered in the Outcomes window in "Specifying a Task Outcome" on page 15-16 display in this list.

   ■ Consensus Percentage
   Select a percentage value required for the outcome of this task to take effect; for example, a majority vote (51) or a unanimous vote (100). For example, assume there are two possible outcomes (ACCEPT and REJECT) and five subtasks. If two subtasks are accepted and three are rejected, and the required acceptance percentage is 50%, the outcome of the task is rejected.

2. Specify additional group voting details:

   ■ Immediately trigger voted outcome when minimum percentage is met
   If selected, the outcome of the task can be computed early with the outcomes of the completed subtasks, enabling the pending subtasks to be withdrawn. For example, assume four users are assigned to act on a task, the default outcome is APPROVE, and the consensus percentage is set at 50. If the first two users approve the task, the third and fourth users do not need to act on the task, since the consensus percentage value has already been satisfied.

   ■ Wait until all votes are in before triggering outcome
   If selected, the workflow waits for all responses before an outcome is initiated.

Specifying a Time Limit for Acting on a Task

1. Click the + sign to expand the Advanced section shown in Figure 15–14.

2. Select Limit allocated duration to

3. Specify the amount of time a user or group receives to act on a task. If the user or group does not act in the time specified, the global escalation and renewal policies that you set in the Expiration and Escalation Policy section (known as the routing slip level) of the Human Task editor are applied. For example, if the global policy is set to escalate the task and this participant does not act in the duration provided, the task is escalated to the manager or another user, as appropriate.
Configuring the Management Chain Participant Type

Figure 15–15 displays the Management Chain window. This participant type routes tasks for approval to multiple users in a management chain hierarchy. You specify the task participants as a management chain list or a list of users.

For example, a purchase order is assigned to a manager. If the manager approves the order, it is assigned to their manager. If that manager approves it, it is assigned to their manager, and so on until three managers approve the order. If any of the managers reject the request or the request expires, the order is rejected.

Figure 15–15 Add Participant Type — Management Chain

1. Enter a recognizable label for this participant in the Label field. This label must be unique within this workflow (for example, Approval Manager, Primary Reviewers, and so on).

Instructions for configuring the following subsections of the Add Participant Type - Management Chain window are listed in Table 15–5:
Assigning Participants to the Management Chain Task

1. Select a method for assigning a user or group to participate in this task.
   - **By name**
     Enter a user or group name or click the first icon (flashlight) to the right of the field to display a window for selecting a user or group configured through the identity service. The identity service enables user authorization and the lookup of user properties, roles, group memberships, and privileges. User information is obtained from Java AuthoriZatioN (JAZN) or an LDAP server such as Oracle Internet Directory. You can use wild cards (*) to search for IDs.
   - **By expression**
     Dynamically assign this task to a user (for example, jcooper) or group (for example, administrators) by clicking the icon to the right of the field to display the Expression Builder window. Users who are members of a group are assigned this task. For a user to act on a task assigned to a group, they must first claim the task in the Oracle BPEL Worklist Application during run time.

See Also: "Assigning Participants to the Single Approver Task" on page 15-25 for rules to follow when specifying assignees with XPath expressions

Bypassing a Task Participant

1. Select the Specify skip rule check box if you want the user or group to be bypassed if a specific condition is satisfied. This action displays an icon for accessing the Expression Builder window for building a condition. For example, if a user submits a business trip expense report that is below a specific amount, no approval is required by their manager. The expression must evaluate to a Boolean value.

See Also: "Bypassing a Task Participant" on page 15-26 for an example of a valid XPath expression for skipping a participant

Specifying the Number of Approvers

---

<table>
<thead>
<tr>
<th>For This Subsection...</th>
<th>See...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requires management chain approval of one of the participants below</td>
<td>Assigning Participants to the Management Chain Task on page 15-31</td>
</tr>
<tr>
<td>Specify skip rule</td>
<td>Bypassing a Task Participant on page 15-31</td>
</tr>
<tr>
<td>Maximum Number of Chain Levels Up</td>
<td>Specifying the Number of Approvers on page 15-31</td>
</tr>
<tr>
<td>Highest Title of Approver</td>
<td></td>
</tr>
<tr>
<td>Limit allocated duration to</td>
<td>Specifying a Time Limit for Acting on a Task on page 15-32</td>
</tr>
<tr>
<td>Allow this participant to invite other participants</td>
<td>Inviting Additional Participants to a Task on page 15-32</td>
</tr>
</tbody>
</table>

Table 15–5 Add Participant Type - Management Chain

<table>
<thead>
<tr>
<th>Requires management chain approval of one of the participants below</th>
<th>Assigning Participants to the Management Chain Task on page 15-31</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specify skip rule</td>
<td>Bypassing a Task Participant on page 15-31</td>
</tr>
<tr>
<td>Maximum Number of Chain Levels Up</td>
<td>Specifying the Number of Approvers on page 15-31</td>
</tr>
<tr>
<td>Highest Title of Approver</td>
<td></td>
</tr>
<tr>
<td>Limit allocated duration to</td>
<td>Specifying a Time Limit for Acting on a Task on page 15-32</td>
</tr>
<tr>
<td>Allow this participant to invite other participants</td>
<td>Inviting Additional Participants to a Task on page 15-32</td>
</tr>
</tbody>
</table>
1. Specify the following task routing parameters. When both parameters are specified, task routing is determined by both parameters. The routing continues until one of these parameters is satisfied. If you select By Expression from the lists below, you can dynamically specify the details by clicking the icon to the right of the field to display the Expression Builder window.

- **Maximum Number of Chain Levels Up**
  Enter a value for the number of levels in the management chain to include in this task. For example, if set to 2 and the task is initially assigned to user jcooper, both the user jstein (manager of jcooper) and the user wfaulk (manager of jstein) are included in the list (apart from jcooper, the initial assignee). This is a mandatory field.

- **Highest Title of Approver**
  Select the title of the last (highest) user in the management chain. The title is retrieved from the identity service.

### Specifying a Time Limit for Acting on a Task
1. Click the + sign to expand the Advanced section shown in Figure 15–15.
2. Select Limit allocated duration to.
3. Specify the amount of time a user or group receives to act on a task. If the user or group does not act in the time specified, the global escalation and renewal policies that you set in the Expiration and Escalation Policy section (known as the routing slip level) of the Human Task editor are applied. For example, if the global policy is set to escalate the task and this participant does not act in the duration provided, the task is escalated to the manager or another user, as appropriate.

See Also: "Escalating, Renewing, or Ending the Task" on page 15-39 for instructions on setting the global escalation and renewal policies in the Expiration and Escalation Policy section of the Human Task editor.

### Inviting Additional Participants to a Task
1. Click the + sign to expand the Advanced section (if not already expanded).
2. Select Allow this participant to invite other participants if you want this task assignee to invite other participants into the workflow before routing it to the next assignee in this workflow. For example, assume the approval workflow goes from James Cooper to John Steinbeck. If this option is checked, James Cooper can decide to first route it to Irving Stone before it goes to John Steinbeck.

**Note:** For the management chain participant type, the additional participants can be invited only by the last user in the management chain.

### Configuring the Sequential List of Approvers Participant Type

Figure 15–16 displays the Sequential List of Approvers window.

This enables you to create a list of sequential participants for a workflow. For example, if you want a document to be reviewed by John, Mary, and Scott in sequence, use this participant type. This is similar to the management chain participant type, except that
with that type, the users are part of an organization hierarchy. For the sequential list of
approvers participant type, they can be any list of users or groups.

Figure 15–16  Add Participant Type — Sequential List of Approvers

1. Enter a recognizable label for this participant in the Label field. This label must be
   unique within this workflow (for example, Approval Manager, Primary Reviewers, and so on).

Instructions for configuring the following subsections of the Add Participant Type —
Sequential List of Approvers window are listed in Table 15–6.

Table 15–6  Add Participant Type — Sequential List of Approvers

<table>
<thead>
<tr>
<th>For This Subsection...</th>
<th>See...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requires sequential approval of all participants below</td>
<td>Assigning Participants to the Sequential List of Approvers Task on page 15-33</td>
</tr>
<tr>
<td>Specify skip rule</td>
<td>Bypassing a Task Participant” on page 15-34</td>
</tr>
<tr>
<td>Limit allocated duration to</td>
<td>Specifying a Time Limit for Acting on a Task” on page 15-34</td>
</tr>
<tr>
<td>Allow this participant to invite other participants</td>
<td>“Inviting Additional Participants to a Task” on page 15-34</td>
</tr>
</tbody>
</table>

Assigning Participants to the Sequential List of Approvers Task

1. Select a method for assigning a user or group to participate in this task.

   • By name
Enter a user or group name or click the first icon (flashlight) to the right of the field to display a window for selecting a user or group configured through the identity service. The identity service enables user authorization and the lookup of user properties, roles, group memberships, and privileges. User information is obtained from Java AuthoriZation (JAZN) or an LDAP server such as Oracle Internet Directory. You can use wild cards (*) to search for IDs.

- **By expression**
  
  Dynamically assign this task to a user (for example, jcooper) or group (for example, administrators) by clicking the icon to the right of the field to display the Expression Builder window. Users who are members of a group are assigned this task. For a user to act on a task assigned to a group, they must first claim the task in the Oracle BPEL Worklist Application during run time.

**See Also:** "Assigning Participants to the Single Approver Task" on page 15-25 for rules to follow when specifying assignees with XPath expressions

**Bypassing a Task Participant**

1. Select the Specify skip rule check box if you want the user or group to be bypassed if a specific condition is satisfied. This action displays an icon for accessing the Expression Builder window for building a condition. For example, if a user submits a business trip expense report that is below a specific amount, no approval is required by their manager. The expression must evaluate to a Boolean value.

**See Also:** "Bypassing a Task Participant" on page 15-26 for an example of a valid XPath expression for skipping a participant

**Specifying a Time Limit for Acting on a Task**

1. Click the + sign to expand the Advanced section shown in Figure 15-16.
2. Click Limit allocated duration to.
3. Specify the amount of time a user or group receives to act on a task. If the user or group does not act in the time specified, the global escalation and renewal policies that you set in the Expiration and Escalation Policy section (known as the routing slip level) of the Human Task editor are applied. For example, if the global policy is set to escalate the task and this participant does not act in the duration provided, the task is escalated to the manager or another user, as appropriate.

**See Also:** "Escalating, Renewing, or Ending the Task" on page 15-39 for instructions on setting the global escalation and renewal policies in the Expiration and Escalation Policy section of the Human Task editor

**Inviting Additional Participants to a Task**

1. Click the + sign to expand the Advanced section (if not already expanded).
2. Select Allow this participant to invite other participants if you want this task assignee to invite other participants into the workflow before routing it to the next assignee in this workflow. For example, assume the approval workflow goes from
James Cooper to John Steinbeck. If this option is checked, James Cooper can decide to first route it to Irving Stone before it goes to John Steinbeck.

### Note:
For the sequential list of approvers participant type, the additional participants can be invited only by the last user in the management chain.

**Configuring the FYI Assignee Participant Type**

Figure 15–17 displays the FYI Assignee window.

This participant type is used when a task is sent to a user, but the business process does not wait for a user response; it just continues. FYI assignees cannot directly impact the outcome of a task, but in some cases can provide comments or add attachments.

For example, a magazine subscription is due for renewal. If the user does not cancel the current subscription before the expiration date, the subscription is renewed. This user is reminded weekly until the request expires or the user acts on it.

![Figure 15–17 Add Participant Type — FYI Assignee](image)

1. Enter a recognizable label for this participant in the **Label** field. This label must be unique within this workflow (for example, Approval Manager, Primary Reviewers, and so on).

Instructions for configuring the following subsections of the Add Participant Type - FYI Assignee window are listed in Table 15–7:

<table>
<thead>
<tr>
<th>For This Subsection...</th>
<th>See...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Send an FYI copy of this task to all participants below</td>
<td>“Assigning Participants to the FYI Assignee Task” on page 15-35</td>
</tr>
<tr>
<td>Share attachments and comments</td>
<td>“Sharing Attachments and Comments with Task Participants” on page 15-36</td>
</tr>
</tbody>
</table>

**Assigning Participants to the FYI Assignee Task**
1. Select a method for assigning a user or group to participate in this task.

   ■ **By name**
   
   Enter a user or group name or click the first icon (flashlight) to the right of the field to display a window for selecting a user or group configured through the identity service. The identity service enables user authorization and the lookup of user properties, roles, group memberships, and privileges. User information is obtained from Java AuthoriZatioN (JAZN) or an LDAP server such as Oracle Internet Directory. You can use wild cards (*) to search for IDs.

   ■ **By expression**
   
   Dynamically assign this task to a user (for example, jcooper) or group (for example, administrators) by clicking the icon to the right of the field to display the Expression Builder window. Users who are members of a group are assigned this task. For a user to act on a task assigned to a group, they must first claim the task in the Oracle BPEL Worklist Application during run time.

   **See Also:** "Assigning Participants to the Single Approver Task" on page 15-25 for rules to follow when specifying assignees with XPath expressions

**Sharing Attachments and Comments with Task Participants**

1. Select the **Share attachments and comments** check box if you want all group voters or workflow participants to share comments and attachments for this task. This information typically displays in the footer region of the Oracle BPEL Worklist Application.

**Configuring the External Routing Service Participant Type**

Figure 15–18 displays the External Routing Service window.

This participant type enables you to configure an external routing service that dynamically determines the participants in the workflow. If this participant type is specified, all other participant types are ignored. It is assumed that the external routing service provides a list of participant types (single approver, list of approvers, group vote, and so on) at run time to determine the routing of the task.
Task 1: Creating the Human Task Definition with the Human Task Editor

1. Enter a recognizable label for this participant in the Label field. This label must be unique within this workflow (for example, Approval Manager, Primary Reviewers, and so on).

Specifying a Class Name

1. Enter the fully qualified class file name or click the flashlight icon to select the name (for example, the org.mycompany.tasks.RoutingService class name). This class must implement the oracle.bpel.services.workflow.task.IAssignmentService interface.
2. Click the + sign to add name and pair value parameters that can be passed to the external service.

See Also: “Dynamically Assigning Task Participants with the Assignment Service” on page 15-118 for details about using this interface

Allowing All Participants to Invite Other Participants

After you configure a participant type and are returned to the Human Task editor, the Allow all participants to invite other participants check box is enabled, as shown in Figure 15-19.

Figure 15–19 Human Task Editor — Assignment and Routing Policy Section
Task 1: Creating the Human Task Definition with the Human Task Editor

This check box is the equivalent of the Adhoc workflow pattern of previous BPEL releases. This applies when there is at least one participant. In this case, each user selects users or groups as the next assignee when approving the task.

1. If you want this task assignee to invite other participants into the workflow before routing it to the next assignee in this workflow, select the Allow all participants to invite other participants check box.

Abruptly Completing a Condition

After you configure a participant type and are returned to the Human Task editor, the Enable abrupt completion condition check box is enabled, as shown in Figure 15–19.

1. If you want to specify conditions under which to complete the task early, regardless of the other participants in the workflow, select the Enable abrupt completion condition check box.

   The Abrupt Completion Details window appears.

   For example, assume an expense report goes to the manager, and then the director. If the first participant (manager) rejects it, you can end the workflow without sending it to the next participant (director).

   There are two methods for specifying the abrupt completion of a task:

   - Outcomes
   - XPath expression routing condition

   If outcomes are specified, any time the selected task outcome occurs, the task completes. If both outcome and routing condition are specified, the workflow service performs a logical OR on the two.

2. Select appropriate outcomes and click the > button. To select all, click the >> button.

3. Click the icon to the right of the Routing Condition field to display the Expression Builder window for dynamically creating a condition under which to complete this task early. For example, if a user submits a business trip expense report that is below a specific amount, no approval is required by their manager.

4. Click OK to return to the Human Task editor.
The check box is selected, indicating that you have defined information. You can click the icon to the right of the Enable abrupt completion condition check box to edit this information.

**Escalating, Renewing, or Ending the Task**

Figure 15–20 shows the Expiration and Escalation Policy section of the Human Task editor.

You can specify expiration duration of a task in this global policy section (also known as the routing slip level). If expiration duration is specified at the routing slip level instead of at the participant type level, then this duration is the expiration duration of the task across all the participants. However, if you specify expiration duration at the participant type level (through the Limit allocated duration to field), then those settings take precedence over settings specified in the Expiration and Escalation Policy section (routing slip level).

Figure 15–20 Human Task Editor — Expiration and Escalation Policy Section

---

Overview or Escalation and Expiration Policy

This section provides an overview of how specifying the expiration duration at this level makes this setting the expiration duration of the task across all the participants.

For example, participant LoanAgentGroup and participant Supervisor have 3 days to act on the task between them, as shown in Figure 15–21:

Figure 15–21 Expire After Policy

---

If there is no expiration specified at either the participant level or this routing slip level, then that task has no expiration duration.
If expiration duration is specified at any of the participant’s level, then for that participant the participant expiration duration is used. However, the global expiration duration is still used for the participants that do not have participant level expiration duration. The global expiration duration is always decremented by the time elapsed in the task.

The policy to interpret the participant level expiration for the participants is described below:

- Management Chain — Each participant in the management chain gets the same expiration duration. The duration is not for all the assignments resulting from this assignment. If the task expires at any of the assignments in the management chain, the task expires and the escalation and renewal policy is applied.

- Sequential list of approvers — Each assignment in the management chain gets the same expiration duration as the one specified in the sequential list of approvers. Note that the duration is not for all the assignments resulting from this assignment. If the task expires at any of the assignments in the management chain, the task expires and the escalation and renewal policy is applied.

- Group vote
  - In a group vote workflow, if the parallel participants are specified as a resource, a routing slip is created for each of the resources. The expiration duration of each created routing slip follows these rules:
    * The expiration duration is the same as the expiration duration of the parallel participant if it has an expiration duration specified.
    * The expiration duration that is left on the task if it was specified at the routing slip level.
    * No expiration duration, otherwise.
  - If parallel participants are specified as routing slips, then the expiration duration for the parallel participants are determined by the routing slip.

**Note:** When the parent task expires in a parallel task, the subtasks are withdrawn if those tasks have not expired or completed.

In the following routing slip sample, participant Loan Agent Group has an expiration duration of 1 day and participant Loan Agent Supervisor does not have any expiration duration on the task, even though an expiration duration is specified at the routing slip level. In this example, the routing slip is treated just as if there were no expiration duration specified at the routing slip level.

```xml
<routingSlip xmlns="http://xmlns.oracle.com/pcbpel/workflow/routingslip">
  <expirationDuration>PT10D </expirationDuration>
  <participants>
    <participant name="Loan Agent 1" expirationDuration="PT2D">
      <resource isGroup="true" type="STATIC">jcooper</resource>
    </participant>
    <participant name="Loan Agent 2">
      <resource isGroup="true" type="STATIC">jstein</resource>
    </participant>
  </participants>
  <managementChain name="Loan Approval Chain">
```

15-40 Oracle BPEL Process Manager Developer's Guide
Table 15–8 demonstrates the expiration policy. Note that the management chain in the above example evaluates to two users — wfaulk and cdickens (manager of wfaulk).

Table 15–8 Expiration Policy

<table>
<thead>
<tr>
<th>Participant</th>
<th>Expiration</th>
<th>Actual Time Taken to Approve</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loan Agent 1 - jcooper</td>
<td>2 days (participant level)</td>
<td>One day</td>
</tr>
<tr>
<td>Loan Agent 2 - jstein</td>
<td>9 days (10 - 1 days) (global level)</td>
<td>One day</td>
</tr>
<tr>
<td>Loan Approval Chain - wfaulk (first user in chain)</td>
<td>2 days (participant level)</td>
<td>One day</td>
</tr>
<tr>
<td>Loan Approval Chain - cdickens (second user in chain)</td>
<td>2 days (participant level)</td>
<td>One day</td>
</tr>
<tr>
<td>Reviewer - sfitzger</td>
<td>6 days (10 - 4 days) (global level)</td>
<td>One day</td>
</tr>
</tbody>
</table>

1. Select an escalation and expiration policy. You can enter a fixed time or a dynamic time by clicking the icon to the right of the By Expression field to display the Expression Builder window.

Never Expire Policy

1. If you never want the task to expire, select Never Expire from the list shown in Figure 15–20 on page 15-39.

Expire After Policy

1. If you want the task to expire, select Expire after from the list shown in Figure 15–20 on page 15-39.

2. Specify the maximum time period for the task to remain open.

When the task expires, either the escalation policy or the renewal policy at the routing slip level is applied. If neither is specified, the task expires. The expiration policy at the routing slip level is common to all the participants.

The expiration policy for parallel participants is interpreted as follows.

- If parallel participants are specified as resources in parallel elements, there is no expiration policy for each of those participants.
- If parallel participants are specified as routing slips, then the expiration policy for the routing slip applies to the parallel participants.
1. If you want to extend the expiration period when the user does not respond within the allotted time, select **Renew after** from the list shown in Figure 15-20 on page 15-39.

2. Specify the maximum number of times to continue renewing this task. The renewal policy specifies the number of times the task can be renewed on expiration and the renewal duration. In Figure 15–23, when the task expires, it is renewed at most 3 times. It does not matter if the task expired at the LoanAgentGroup participant or the Supervisor participant.

**Escalate After Policy**

1. If you want to escalate the task (for example, to the user’s manager) if the user does not respond within the allotted time, select **Escalate after** from the list shown in Figure 15–20 on page 15-39.
2. Specify the following additional values:
   - **Maximum Escalation Levels**
     Number of management levels to which to escalate the task
   - **Highest Approver Title**
     The title of the highest approver (for example, self, manager, director, or CEO).

   The escalation policy specifies the number of times the task can be escalated on expiration and the renewal duration. In Figure 15–24, when the task expires, it is escalated at most 3 times. It does not matter if the task expired at the LoanAgentGroup participant or the Supervisor participant.

   ![Figure 15–24 Escalate After Policy](image)

   **Specifying Participant Notification Preferences**

   Figure 15–25 shows the Notification Settings section of the Human Task editor (when fully expanded).

   Notifications indicate when a user is assigned a task or informed that the status of the task has changed. Notifications can be sent through e-mail, voice message, fax, pager, or SMS. Notifications are sent to different types of participants for different actions.

   Notifications are configured by default with default messages. For example, a notification message is sent to indicate that a task has completed and closed. You can create your own or modify existing configurations.
Task 1: Creating the Human Task Definition with the Human Task Editor

Figure 15–25 Human Task Editor — Notification Settings Section

1. Click the + sign to expand the Notification Settings section (displays as shown in Figure 15–25).

Instructions for configuring the following subsections of the Notification Settings section are listed in Table 15–9.

Table 15–9 Human Task Editor — Notification Settings Section

<table>
<thead>
<tr>
<th>For This Subsection...</th>
<th>See...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task Status Recipient</td>
<td>&quot;Notifying Recipients of Changes to Task Status&quot; on page 15-44</td>
</tr>
<tr>
<td>Notification Header</td>
<td>&quot;Editing the Notification Message&quot; on page 15-45</td>
</tr>
<tr>
<td>Reminders</td>
<td>&quot;Setting Up Reminders&quot; on page 15-45</td>
</tr>
<tr>
<td>Make notifications secure (exclude details)</td>
<td>&quot;Securing Notifications, Making Messages Actionable, and Sending Attachments&quot; on page 15-46</td>
</tr>
</tbody>
</table>

Make e-mail messages actionable
Send task attachments with email notifications

See Also: "Notifications from Workflow Services" on page 15-80

Notifying Recipients of Changes to Task Status

Three default status types display in the Task Status column: Assign, Complete, and Error. You can select other status types for which to receive notification messages.

1. Click a type in the Task Status column to display the complete list of task types:

   - Assign—when the task is assigned to users or a group. This action captures the following actions:
     - Task is assigned to a user
     - Task is assigned to a new user in a sequential list of approvers workflow
     - Task is renewed
     - Task is delegated
     - Task is reassigned
     - Task is escalated
Task 1: Creating the Human Task Definition with the Human Task Editor

1. Information for a task is submitted
- Complete
- Error
- Expire
- Request Info
- Update Outcome
- Suspend
- Withdraw

2. Select a task status type.
Notifications can be sent to users involved in the task in various capacities. This includes when the task is assigned to a group, each user in the group is sent a notification if there is no notification endpoint available for the group.

3. Click an entry in the Recipient column to display a list of possible recipients for the notification message.
- Assignees—the users or groups to whom the task is currently assigned
- Initiator—the user who created the task
- Approvers—the users who have approved the task so far. This applies in a sequential list of approvers participant type where multiple users have approved the task and a notification must be sent to all of them.
- Owner—the task owner

See Also: “Configuring the Notification Channel” on page 15-81

Editing the Notification Message
A default notification message is available for delivery to the selected recipient. If you want, you can modify the default message text.

1. Click the icon in the Notification Header column to modify the default notification message.

The Edit Notification Message window appears.

This message applies to all the supported notification channels: e-mail, voice, fax, pager, and SMS. E-mail and fax messages can also include the worklist task detail
defined in this message. The channel by which the message is delivered is based upon the notification preferences you specify.

2. Modify the message wording as necessary.

3. Click OK to return to the Human Task editor.

See Also: "Notifications from Workflow Services" on page 15-80 for notification preference details

Setting Up Reminders
You can send task reminders, which can be based on the time the task was assigned to a user or the expiration time of a task. The number of reminders and the interval between the reminders can also be configured.

1. Select the number of reminders to send from the Remind list.

2. If you selected to remind the assignee one, two, or three times, select the interval between reminders, and whether to send the reminder before or after the assignment.

See Also: "Sending Reminders" on page 15-85

Securing Notifications, Making Messages Actionable, and Sending Attachments
You can perform additional notification tasks in this section.

1. Select the corresponding check box for functionality you want to use.

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Make notifications secure (exclude details)</td>
<td>Select to make the notification message secure. If selected, a default notification message is used. There are no HTML worklist task details, attachments, or actionable links in the e-mail. Only the task number is in the message. See Also: &quot;Sending Secure Notifications&quot; on page 15-85</td>
</tr>
<tr>
<td>Make e-mail messages actionable</td>
<td>Select to make e-mail notifications actionable. This enables you to perform task actions through e-mail. See Also: &quot;Sending Actionable E-mails&quot; on page 15-83 for additional configuration details</td>
</tr>
<tr>
<td>Send task attachments with e-mail</td>
<td>Select to send task attachments with the notification message. See Also: &quot;Sending Inbound and Outbound Attachments&quot; on page 15-84</td>
</tr>
</tbody>
</table>
Task 1: Creating the Human Task Definition with the Human Task Editor

Specifying Advanced Settings

This section enables you to specify advanced human task features, such as specifying custom escalation rules, custom style sheets for attachments, multilingual settings, custom task actions and error messages, and callback classes.

Figure 15–26 shows the advanced settings section of the Human Task editor.

**Table 15–10** Advanced Settings Sections

<table>
<thead>
<tr>
<th>Section</th>
<th>See...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specify escalation rule</td>
<td>“Specifying Escalation Rules” on page 15-47</td>
</tr>
<tr>
<td>Specify WordML Stylesheet for attachments</td>
<td>“Specifying WordML Style Sheets for Attachments” on page 15-48</td>
</tr>
<tr>
<td>Specify stylesheet for attachments</td>
<td>“Specifying Style Sheets for Attachments” on page 15-48</td>
</tr>
<tr>
<td>Specify multilingual settings</td>
<td>“Specifying Multilingual Settings” on page 15-48</td>
</tr>
<tr>
<td>Override default system actions</td>
<td>“Overriding Default System Actions” on page 15-49</td>
</tr>
<tr>
<td>Override default exception management</td>
<td>“Overriding Default Exception Management” on page 15-51</td>
</tr>
<tr>
<td>Specify callback class on task status</td>
<td>“Specifying Callback Classes on Task Status” on page 15-51</td>
</tr>
<tr>
<td>Allow task and routing customization in BPEL callbacks</td>
<td>“Allowing Task and Routing Customization in BPEL Callbacks” on page 15-51</td>
</tr>
</tbody>
</table>

Specifying Escalation Rules

This option allows a custom escalation rule to be plugged in for a particular workflow. For example, to assign the task to a current user’s department manager on task expiration, you can write a custom task escalation function, register it with the workflow service, and use that function in task definitions.

The default escalation rule is to assign a task to the manager of the current user. To add a new escalation rule, follow the steps below:
Task 1: Creating the Human Task Definition with the Human Task Editor

1. Implement interface
   oracle.bpel.services.workflow.assignment.dynamic.IDynamicTask
   EscalationFunction. This implementation has to be available in the classpath
   for the server.

2. Change the file SOA_Oracle_
   Home\bpel\system\services\config\wf-dynamic-assign-cfg.xml to
   add a new function:
   
   `<dynamicAssignmentFunctions>
   
   <function name="MANAGERS_MANAGER"
   classpath="oracle.bpel.services.workflow.assignment.dynamic.patterns.
   TaskEscalationManagersManager">
   </function>
   
   </dynamicAssignmentFunctions>

3. Enter the function name as defined in the wf-dynamic-assign-cfg.xml file
   for the escalation rule in the Specify Escalation Rule field.

   See Also: "Custom Escalation Function" on page 15-122

Specifying WordML Style Sheets for Attachments
This option allows dynamic creation of Microsoft Word documents for the purpose
of sending them as e-mail attachments using a WordML XSLT stylesheet. The XSLT
stylesheet is applied on the task document.

1. Click the flashlight icon to select a WordML style sheet as an attachment.

   See Also: Oracle Application Server Developer’s Guide for Microsoft
   Office Interoperability for specific details

Specifying Style Sheets for Attachments
This option allows creation of e-mail attachments using an XSLT stylesheet. The XSLT
stylesheet is applied on the task document.

1. Click the flashlight icon to select a stylesheet as an attachment.

Specifying Multilingual Settings
You can specify resource bundles for displaying task details in different languages in
the Oracle BPEL Worklist Application. Resource bundles are supported for the
following task details.

- Displaying the value for task outcomes in plain text or with the message(key)
  format

- Displaying the XML element and attributes names in the payload display of the
  Oracle BPEL Worklist Application. The key name in the resource bundle must be
  the same as the name of the XML element and attributes for internationalization of
  XML element names in the Oracle BPEL Worklist Application.

- Making e-mail notification messages available in different languages. At run time,
  specify the XPath extension function
  hwf:getTaskResourceBundleString(taskId, key, locale?) to obtain
  the internationalized string from the specified resource bundle. The locale of the
  notification recipient can be retrieved with the function
  hwf:getNotificationProperty(propertyName).
1. Click Configure Resource.
   The Resource Details window appears.

   ![Resource Details Window]

2. Enter the name of the resource used in the resource bundle.
3. Click the flashlight icon to select the JAR or ZIP resource bundle file to use. The resource bundle can be part of your BPEL suitcase.
4. Click OK to return to the Human Task editor.

   **See Also:** "Configuring Messages in Different Languages" on page 15-83

**Overriding Default System Actions**

The actions performed from the Oracle BPEL Worklist Application are common to all business processes. However, you can restrict some actions in a particular business process.

For example, assume that in a loan approval process, the business requirement is to never suspend a loan application. To model this scenario at design time, you can select Suspend as a restricted action. When an action is selected as restricted, the Oracle BPEL Worklist Application does not display the action for this task.

By default, these actions are available on all tasks based on the user’s privileges. The task owner or bpeladmin administrator can always perform any of these actions on processes they own.

1. Click Configure Actions.
2. Select the system actions allowed on a task. By default, all are selected and available (unrestricted).
The following system actions can be restricted by unselecting them:

- **Suspend** — Enables task owners (or users with the `BPMWorkflowSuspend` privilege) to put a workflow temporarily on hold. Task expiration and escalation do not apply until the workflow is resumed. No actions are permitted on a suspended task (except resume and withdraw).

- **Push back** — Sends the task one level back in the workflow. For example, assume the task was routed to the `LoanAgentGroup` and then to `jstein`. If `jstein` now pushes the task back, it goes back to the `LoanAgentGroup`.

- **Renew** — If a task is about to expire, a task assignee can renew the task and request more time to perform the task. This operation is not allowed if the process designer has restricted task renewal on the workflow.

- **Skip current assignment** — Skips the current assignment and moves to the next assignment or picks the outcome as set by the previous approver if there are no more assignees.

- **Adhoc Route** — Enables a user to enter an outcome and then route the task in an adhoc fashion to the next user who must review the task.

- **Request Information** — Any workflow participant can request information from the task initiator or any of the prior approvers of the task. When the requested information is submitted, the task is assigned to the user who requested the information.

- **Delegate** — Any workflow participant can delegate the task to another user. In this case, the other user is acting on behalf of the current assignee. When the task is delegated, it resides on both users’ worklists until the original assignee or the delegated person acts on it.

- **Reassign** — Enables the current assignee of the task to transfer it to another user or group. In this case, the task is moved from the worklist of the current assignee to the new assignee.

- **Escalate** — Escalates a task to their manager for further action.

- **Withdraw** — Enables the task initiator to withdraw any pending task if they no longer want to send it through the workflow. A task owner can also withdraw a task on behalf of the initiator. When a task is withdrawn, the
business process is called back with the state attribute of the task set to Withdrawn.

3. Click OK to return to the Human Task editor.

Overriding Default Exception Management

Tasks can error due to incorrect assignments. Incorrect assignments can occur for any of the following reasons:

- Invalid assignees — The task assignee user or group is not a valid user in the system.
- Invalid dynamic assignment — When assignees are specified to be dynamic, the dynamic XPath expression is not evaluated.

In the above cases, the task is marked as errored. By default, the life cycle of the task is completed with that action.

During modeling of workflow tasks, you can specify error assignees for the workflow. If error assignees are specified, they are evaluated and the task is assigned to them. The error assignee can perform one of the following actions:

- Adhoc route — route the task to the actual users assigned to the task. Adhoc route allows the task to be routed to users in sequence, parallel, and so on.
- Reassign — reassign the task to the actual users assigned to this task
- Error task — indicate that this task cannot be rectified.

If there are any errors in evaluating the error assignees, the task is marked as errored. This window enables you to specify the users or groups to whom the task is assigned if an error in assignment has occurred.

1. Click Configure Assignment.
2. Select the error assignees.

Specifying Callback Classes on Task Status

You can register callbacks for the workflow service to call during the life cycle of a task. The callback class has to implement the interface oracle.bpel.services.workflow.task.IRoutingSlipCallback. Make the callback class available in the classpath of the server.
1. Click Configure Callbacks.

2. Click the + sign to add a callback to the table. A callback named OnAssigned is automatically added to the Callback column.

3. Click OnAssigned to display a list of additional callback values to select for this column.

   The following callbacks are available:
   - onCompleted — This callback is invoked when the task is completed, expired, withdrawn, or errored.
   - onAssigned — This callback is invoked when the task is assigned to a new set of assignees due to the following actions:
     - outcome update
     - skip current assignment
     - override routing slip
   - onUpdated — This callback is invoked for any other update to the task that does not fall in the onTaskComplete or onTaskAssigned callback. This includes updates on a task due to request for information, submit information, escalation, reassign, and so on.
   - onSubtaskUpdated — This callback is invoked for any update to a subtask.

4. Click Java in the Type column to display a list of additional values for this column.

5. Click the empty field in the Value column to enter a value. The value is the complete class name of the Java class that implements oracle.bpel.services.workflow.task.IRoutingSlipCallback.

6. Click OK.

Allowing Task and Routing Customization in BPEL Callbacks

The Allow task and routing customization in BPEL callbacks check box must be selected if you select the check box of the same name on the Human Task - Advanced tab shown in Figure 15–28 on page 15-58. Selecting both check boxes updates the metadata for callbacks.

See Also: "Allowing Task and Routing Customizations in BPEL Callbacks" on page 15-59 for details on using callbacks
Exiting the Human Task Editor and Saving Your Changes

You can save your human task changes at any time. The task can be re-edited at a later time by clicking the metadata task configuration .task file in the Application Navigator.

1. Select Save from the File main menu or click the X sign to close the .task metadata task configuration file.

2. If you click the X sign, select Yes when prompted to save your changes.

Task 2: Associating the Human Task with a BPEL Process

You must associate the .task file that consists of the human task settings with a BPEL process. When association is complete, a Task Service partner link is created. The Task Service exposes the operations required to act on a task.

The method by which you created the human task indicates if the task is already associated with a BPEL process. Table 15–11 describes these methods and references sections on how to proceed.

Note: Note that regardless of whether you have already associated the human task with a BPEL process, you must still define key human task activity properties, including the task title, task initiator, task priority, and task parameter variables. These tasks are described in “Defining the Human Task Activity Title, Initiator, Priority, and Parameter Variables” on page 15-55 and "Defining the Human Task Activity Advanced Features" on page 15-57.

<table>
<thead>
<tr>
<th>Human Task Creation Method</th>
<th>Then...</th>
<th>See...</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Dragged and dropped a human task activity into the BPEL process.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2. Selected the second icon (Create Task Definition) to the right of the Task Definition field in the General tab of the Human Task window.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

See Also: “Task 1: Creating the Human Task Definition with the Human Task Editor” on page 15-13 for instructions on creating a human task.
Task 2: Associating the Human Task with a BPEL Process

**Associating a Human Worklist Task with a BPEL Process**

1. Select the BPEL process with which to associate the `.task` file of the human task in the Application Navigator.
2. Select Process Activities from the Component Palette.
3. Drag and drop a new Human Task activity into your BPEL process.
   The Add a Human Task window appears.

   **Note:** When you first drag and drop this activity into Oracle JDeveloper, the window is named Add a Human Task. After you finish specifying details on this window and click OK, the name of this window changes to simply Human Task.

4. Click the first icon to the right of the Task Definition field.

   The Choose Task Definition File appears.

5. Select the `.task` file and click Open. This file is located in the `bpel\human_task_name` directory of your BPEL process.

   The `.task` file is added to the Task Definition field.

6. See the following sections to configure the human task activity:
   - Defining the Human Task Activity Title, Initiator, Priority, and Parameter Variables
   - Defining the Human Task Activity Advanced Features

**Opening a Human Task Activity Already Associated with a BPEL Process**

1. Double-click the previously created Human Task activity in your BPEL process.
   The Human Task window appears.

2. Click the third icon to the right of the Task Definition field to open the human worklist task you previously created.

3. See the following sections to configure the human task activity:
   - Defining the Human Task Activity Title, Initiator, Priority, and Parameter Variables
   - Defining the Human Task Activity Advanced Features
Defining the Human Task Activity Title, Initiator, Priority, and Parameter Variables

Figure 15–27 shows the General tab.

![Human Task — General Tab](image)

The General tab of the Human Task activity enables you to perform the tasks shown in Table 15–12:

<table>
<thead>
<tr>
<th>For this Field...</th>
<th>See...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task Title</td>
<td>“Specifying the Task Title” on page 15-55</td>
</tr>
<tr>
<td>Initiator</td>
<td>“Specifying the Task Initiator and Task Priority” on page 15-55</td>
</tr>
<tr>
<td>Priority</td>
<td></td>
</tr>
<tr>
<td>Task Parameters</td>
<td>“Specifying Task Parameters” on page 15-56</td>
</tr>
</tbody>
</table>

Specifying the Task Title

1. Enter the task title in the Task Title field through one of the following methods. This is a mandatory field. Your entry in this field overrides the task title you entered in the Title field of the Human Task editor described in “Specifying a Task Title and Priority” on page 15-16. The title displays the task in the Oracle BPEL Worklist Application during run time.
   - Enter the title manually.
   - Click the icon to the right of the field to display the Expression Builder window to dynamically create the title.

You can also mix static text and dynamic expressions in the same title. To include dynamic text, place your cursor at the appropriate point in the text and click the icon on the right to invoke the Expression Builder window.

See Also: “Assigning Input and Output Parameters for the Human Task” on page 15-92 for an example of specifying the task title name

Specifying the Task Initiator and Task Priority

1. Enter the initiator (for example, jcooper) or click the icon to the right of the Initiator field to display the Expression Builder window for dynamically specifying an initiator. This field is optional.
The initiator is the user who initiates a task. The initiator can view their created tasks from the Oracle BPEL Worklist Application and perform specific tasks defined in the System Action Details window, such as withdrawing or suspending a task. If not specified, the initiator defaults to the task owner specified on the Advanced tab of the Human Task window. The initiator defaults to bpeladmin if a task owner is also not specified.

2. Select a priority value between 1 (the highest) and 5 from the Priority list. This field is provided for user reference and does not make this task a higher priority during run time. The priority can be used to sort tasks in the Oracle BPEL Worklist Application. This priority value overrides the priority value you select in the Priority list of the Human Task editor.

**See Also:** "Specifying a Task Title and Priority" on page 15-16 for instructions on specifying the priority in the Human Task editor

**Specifying Task Parameters**

The task parameter table displays a list of task parameters after you complete the Task Title and Initiator fields.

1. Click the flashlight in the BPEL Variable column to map the task parameter to the BPEL variable. You must map only the task parameters that carry input data. For output data that is filled in from the worklist, you do not need to map the corresponding variables.

   The Task Parameters window appears.

2. Expand the Variables navigation tree and select the appropriate task variable.
Task 2: Associating the Human Task with a BPEL Process

3. Click OK.
   The Human Task window appears as follows.

4. Click Apply.
5. If you want to define advanced features for the human task activity, click the Advanced tab and go to section “Defining the Human Task Activity Advanced Features” on page 15-57. Otherwise, click OK to close the Human Task window.

Defining the Human Task Activity Advanced Features
Figure 15–28 shows the Advanced tab.
Task 2: Associating the Human Task with a BPEL Process

Figure 15–28 Human Task — Advanced Tab

The Advanced tab of the Human Task activity enables you to perform the tasks shown in Table 15–13:

Table 15–13 Human Task - Advanced Tab

<table>
<thead>
<tr>
<th>For this Field...</th>
<th>See...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scope Name</td>
<td>“Specifying a Scope Name and a Global Task Variable Name” on page 15-58</td>
</tr>
<tr>
<td>Global Task Variable Name</td>
<td>“Specifying a Task Owner” on page 15-58</td>
</tr>
<tr>
<td>Owner</td>
<td>“Specifying an Identification Key” on page 15-59</td>
</tr>
<tr>
<td>Identification Key</td>
<td>“Including the Task History of Other Human Tasks” on page 15-59</td>
</tr>
<tr>
<td>Include task history from</td>
<td>“Allowing Task and Routing Customizations in BPEL Callbacks” on page 15-59</td>
</tr>
<tr>
<td>Allow task and routing customization in BPEL callbacks</td>
<td>“Allowing Task and Routing Customizations in BPEL Callbacks” on page 15-59</td>
</tr>
</tbody>
</table>

Specifying a Scope Name and a Global Task Variable Name

You are automatically provided with default scope and global task variable names during human task activity creation. However, you can specify custom names that are used to name the scope and global variable during human task activity creation.

1. Enter the name for the BPEL scope to be generated in the Scope Name field.
   - This BPEL scope encapsulates the entire interaction with the workflow service and BPEL variable manipulation.

2. Enter the global task variable name in the Global Task Variable Name field.
   - This is the name of the BPEL task variable used for the workflow interaction.

Specifying a Task Owner

1. Enter the task owner name in the Owner field or click the icon to the right to use the Expression Builder to dynamically specify the owner of this task.
The task owner can view tasks belonging to business processes they own and perform operations on behalf of any of the task assignees. Additionally, the owner can also reassign, withdraw, or escalate tasks.

If you do not specify a task initiator on the General tab of the Human Task window, it defaults to the owner specified here. If an owner is not specified, it defaults to the bpeladmin administrator.

Specifying an Identification Key
1. Enter an optional identification key value in the Identification Key field.

   The identification key can be used as a user-defined ID for the task. For example, if the task is meant for approving a purchase order, the purchase order ID can be set as the identification key of the task. Tasks can be searched from the Oracle BPEL Worklist Application using the identification key. This attribute has no default value.

Including the Task History of Other Human Tasks
This feature enables one workflow to be continued with another workflow.
1. Select the Include task history from check box to extend a previous workflow task in the BPEL process. Selecting this check box includes the task history, comments, and attachments from the previous task. This provides you with a complete end-to-end audit trail.

   When a workflow task is continued with another workflow, the following information is carried over to the new workflow:
   ■ Task payload and the changes made to the payload in the previous workflow
   ■ Task history
   ■ Comments added to the task in the previous workflow
   ■ Attachments added to the task in the previous workflow

   In the Include task history from list, all existing workflows are listed. Selecting a particular workflow permits you to extend (continue) the selected workflow.

   For example, a hiring process is used to hire new employees. Each interviewer votes to hire or not hire a candidate. If 75% of the votes are to hire, then the candidate is hired; otherwise, the candidate is rejected. If the candidate is to be hired, an entry in the HR database is created and the human resources contact completes the hiring process. The HR contact also needs to see the interviewers and the comments they made about the candidate. This process can be modeled using a group vote for the hiring. If the candidate is hired, a database adapter is used to create the entry in the HR database. After this, a simple workflow can include the task history from the group vote so that the hiring request, history, and interviewer comments are carried over. This simple workflow is assigned to the HR contact.

See Also: “Including the Task History from Other Workflows” on page 15-64

Allowing Task and Routing Customizations in BPEL Callbacks
1. Select the Allow task and routing customizations in BPEL callbacks check box to notify the BPEL process using OnMessage callbacks every time a task is routed to a different user or when the task status changes. You must also select the check
Task 2: Associating the Human Task with a BPEL Process

box of the same name in the Advanced Settings section of the Human Task editor shown in Figure 15–26 on page 15-47 in order to update the metadata for callbacks. In these callbacks, you can call the Task Service to change the routing or update the current assignees. This option impacts the BPEL code generated to interact with the Task Service. If this option is not selected, the client process gets notified only when the task completes or when it expires or errors out.

2. Click OK to close the Human Task window.

3. Go to the Human Task editor for this human task (the .task file).

4. Expand the Advanced Settings section at the bottom of the editor.

5. Click Allow task and routing customization in BPEL callbacks. This check box must be selected to use callbacks. This enables you to update the metadata.

See Also:

- “Allowing Task and Routing Customization in BPEL Callbacks” on page 15-52
- "BPEL Callbacks" on page 15-62

Viewing the Generated Human Task Activity

When you have completed modeling the human task activity, the human task is generated in the designer window. Figure 15–29 shows how a workflow interaction is modeled in Oracle BPEL Process Manager. Figure 15–29 also illustrates the interaction when no BPEL callbacks are modeled. In this case, once a task is complete, the BPEL process is called back with the completed task. No intermediary events are propagated to the BPEL process instance. It is recommended that any user customizations be done in the first assign, AssignTaskAttributes, and that AssignSystemTaskAttributes not be changed.
Task 2: Associating the Human Task with a BPEL Process

Figure 15–29  Workflow Interaction Modeling

- **AssignTaskAttributes**
  - Captures the user-defined attributes of the task such as title, payload, creator, priority, and so on

- **AssignSystemTaskAttributes**
  - Captures the system task attributes such as process id, process version, and so on

- **InitiateTask**
  - Initiates the task by invoking the task service

- **ReceiveCompletedTask**
  - Receives the completed task from the task service

Figure 15–30 shows a workflow interaction in Oracle JDeveloper.
BPEL Callbacks

If intermediary events need to be propagated to the BPEL process instance, select the Allow task and routing customization in BPEL callbacks check box in both the Advanced tab of the Human Task window and the Advanced Settings section of the Human Task editor. When these options are selected, the workflow service invokes callbacks in the BPEL instance during each update of the task. The callbacks are listed in the TaskService.wsdl file and described below:

- **onTaskCompleted** — This callback is invoked when the task is completed, expired, withdrawn, or errored.
- **onTaskAssigned** — This callback is invoked when the task is assigned to a new set of assignees due to the following actions:
  - Outcome update
  - Skip current assignment
  - Override routing slip
- **onTaskUpdated** — This callback is invoked for any other update to the task that does not fall in the onTaskComplete or onTaskAssigned callback. This includes updates on tasks due to request for information, submit information, escalation, reassign, and so on.
Task 2: Associating the Human Task with a BPEL Process

- **onSubTaskUpdated** — This callback is invoked for any update to a subtask.

Figure 15–31 shows how a workflow interaction with callbacks is modeled in Oracle BPEL Process Manager. Once this task is initiated, a while loop is used to receive messages until the task is complete. The while loop contains a pick with four onMessage branches — one for each of the above-mentioned callback operations. The workflow interaction works fine even if nothing is changed in the onMessage branches, meaning that customizations in the onMessage branches are not required.

In this scenario, a workflow context is captured in the BPEL instance. This context can be used for all interaction with the workflow services. For example, if you want to reassign a task if it is assigned to a group, then you need the workflow context for the `reassignTask` operation on the Task Service.

It is recommended that any user customizations be done in the first assign, `AssignTaskAttributes`, and that `AssignSystemTaskAttributes` not be changed.

Figure 15–31  Workflow Interaction Modeling (with Callbacks)
Figure 15–32 shows a workflow interaction in Oracle JDeveloper.

Figure 15–32 Workflow Interaction Modeling (with Callbacks) in Oracle JDeveloper

Including the Task History from Other Workflows
When the task history is included in a workflow, the generated BPEL process described in the previous two sections is similar, with the following differences:

- The BPEL variable from the workflow whose task history is to be included is reused and no new BPEL variable is created.
- The first invoke activity invokes the reinitiate operation instead of the initiate operation.

See Also: "Including the Task History of Other Human Tasks" on page 15-59

Outcome-Based Modeling
In many cases, the outcome of a task determines the flow of the business process. To facilitate modeling of the business logic, when a user task is generated, a BPEL switch activity is also generated with prebuilt BPEL case activities. By default, one case branch is created for each outcome selected during creation of the task. An otherwise branch is also generated in the switch to represent cases when the task is withdrawn, expired, or errored.
Task 2: Associating the Human Task with a BPEL Process

Payload Updates
The task carries a payload in it. If the payload is set from a business process variable, then an assign activity with the name copyPayloadFromTask is created in each of the case and otherwise branches to copy the payload from the task back to its source. If the payload is expressed as other XPath expressions (such as ora:getNodes(...)), then this assign is not created because of the lack of a process variable to copy the payload back. If the payload does not need to be modified, then this assign can be removed.

Case Statements for Other Task Conclusions
By default, the switch activity contains case statements for the outcomes only. The other task conclusions are captured in the otherwise branch. These conclusions are as follows:

- The task is withdrawn
- The task is errored
- The task is expired

If business logic must be added for each of these other conclusions, then case statements can be added for each of the preceding conditions. The case statements can be created as shown in the following BPEL segment. The XPath conditions for the other conclusions in the case activities for each of the preceding cases are shown in bold:

```xml
<switch name="taskSwitch">
  <case condition="bpw:getVariableData('SequentialWorkflowVar1', '/task:task/task:state') = 'COMPLETED' and bpw:getVariableData('SequentialWorkflowVar1', '/task:task/task:conclusion') = 'ACCEPT'">
    <bpelx:annotation>
      <bpelx:pattern>Task outcome is ACCEPT</bpelx:pattern>
    </bpelx:annotation>
  </case>
  <case condition="bpw:getVariableData('SequentialWorkflowVar1', '/task:task/task:state') = 'WITHDRAWN'">
    <bpelx:annotation>
      <bpelx:pattern>Task is withdrawn</bpelx:pattern>
    </bpelx:annotation>
  </case>
  <case condition="bpw:getVariableData('SequentialWorkflowVar1', '/task:task/task:state') = 'EXPIRED'">
    <bpelx:annotation>
      <bpelx:pattern>Task is expired</bpelx:pattern>
    </bpelx:annotation>
  </case>
  <case condition="bpw:getVariableData('SequentialWorkflowVar1', '/task:task/task:state') = 'ERRORED'">
    <bpelx:annotation>
      <bpelx:pattern>Task is errored</bpelx:pattern>
    </bpelx:annotation>
  </case>
</switch>
```
Task 3: Generating the Task Display Form

The task display form defines the display mechanism for the task payload in the Oracle BPEL Worklist Application. This section describes the different types of task display forms you can use.

This section contains the following topics:
- Overview of Task Display Forms
- Selecting a Task Display Form
- Automatically Generating a Simple Task Display Form
- Generating a Custom Task Display Form
- Deploying Task Display Forms
- Creating Custom JSP Forms

Overview of Task Display Forms

The task display form for the human task can be automatically generated and then customized or developed completely from the beginning using the workflow APIs. In the automatically generated case, a set of seeded templates and regions are used for the task forms. There are two methods for generating forms associated with the task definition:

- Automatically generate a simple task form — JSP-based forms that use the standard header, body, and footer template.
- Custom task form — enables you to select one of the existing templates and regions to create a task form. You can also specify which task parameters to display in the form.

When task display forms are generated, a .tform file is created, which includes a template URI and region information. The .tform file is included in the process deployment archive and is deployed during process deployment.

See Also: "Automatically Generating a Simple Task Display Form" on page 15-68 for an example of a .tform file

Selecting a Task Display Form

Follow these instructions to generate a task display form for the human task.

1. Go to the Application Navigator in Oracle JDeveloper.
2. Right-click the folder of the human task for which to create a task display form
   (for this example, ExpenseApproval of the ExpenseRequest BPEL process in
   selected).

   The following menu of selections appears.

3. See the following sections for details about generating the different types of task
   forms:

<table>
<thead>
<tr>
<th>Selection</th>
<th>See...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Auto Generate Simple Task Form</td>
<td>&quot;Automatically Generating a Simple Task Display Form&quot; on page 15-68</td>
</tr>
<tr>
<td>Custom Task Form</td>
<td>&quot;Generating a Custom Task Display Form&quot; on page 15-74</td>
</tr>
</tbody>
</table>

**Preview Release of Task Display Form Support for ADF Data Controls**

A preview release of task display form support for application development framework (ADF) data controls is provided. Very minimal support is provided with this preview release. Note the following limitations:

- There is no support for complex XSDs with recursive elements.
- Task forms generated with ADF data controls cannot be edited.

Follow these procedures to use this preview release:

1. Open an operating system command prompt.
2. Open Oracle JDeveloper in preview mode:

   \[\text{JDev}\_\text{Oracle}\_\text{Home}\bin\jdev.exe -J"-Dpreview_mode=true"\]

   Note that Auto Generate Task Form With ADF Datacontrols now appears as a
   menu option when you right-click the folder of the human task, as shown in Step 2
   of "Selecting a Task Display Form" on page 15-67.

3. Open the \text{SOA}\_\text{Oracle}\_\text{Home}\j2ee\OC4J\_\text{Home}\config\server.xml
   file.

   where \text{OC4J}\_\text{Home} is the name of the OC4J container for your install type:
   - home — for the Oracle Application Server SOA Basic install type
   - OC4J\_SOA — if you accepted the default value for the Oracle Application
     Server SOA Advanced install types

4. Add the following line under the \text{<shared-library}
   name="oracle.bpel.common" version="10.1.3"> section:

   \text{<import-shared-library name="adf.oracle.domain"/>}

5. Restart Oracle Application Server SOA Suite for the changes to take effect.
Automatically Generating a Simple Task Display Form

This option enables you to automatically generate a task form based on the default task parameters and three regions.

1. Select **Auto Generate Simple Task Form** from the list shown in Step 2 on page 15-67.

The default layout is based on the following three region template:

- **Header region** — this region has standard task attributes such as title, priority, created date, assignee, and expiration date. This information is contained in the `Header1.jsp` file.

- **Body region** — this region is generated based on the task parameters. Depending on the XSD used in the task, it is either generated as a list of values or as a table (for repeating items). If you specified the parameter to be modifiable through the worklist on the Add Task Parameter window in Step 2 on page 15-21, it displays as an editable field in the form. Otherwise, the field displays as read-only. The information for this region is contained in the `payload-body.jsp` file and the `payload-body.xml` file. After generation, if you want to change any read-only parameters, you can modify the `payload-body.xml` file.

- **Footer region** — this region has an area for comments, attachments, and a short history of the task routing. This information is contained in the `Footer1.jsp` file.

A `.tform` file is generated. The contents of this file are as follows:

```xml
<?xml version='1.0' encoding='UTF-8'?>
<taskDisplay
    targetNamespace="http://xmlns.companyABC.com/workflow/orderTaskDisplay"
    generateInternationalisedJSP="false"
    xmlns:task="http://xmlns.oracle.com/bpel/workflow/task"
    xmlns="http://xmlns.oracle.com/bpel/workflow/taskDisplay">
    <taskDefinitionId>${domain_id}_${process_id}_${process_revision}_Workflow_Name</taskDefinitionId>
    <applicationName>worklist</applicationName>
    <template>${http_url}/${domain_id}/${process_id}/${process_revision}/Workflow_Name/Template_Name.jsp</template>
    <regions>
        <defaultJSP regionName="Header">
            <jspURI>Header1.jsp</jspURI>
        </defaultJSP>
        <autoGeneratedJSP regionName="Body" editable="true">
            <jspURI>payload-body.jsp</jspURI>
            <messageAttribute editable="false">Workflow_Name ProcessRequest</messageAttribute>
        </autoGeneratedJSP>
        <defaultJSP regionName="Footer">
            <jspURI>Footer1.jsp</jspURI>
        </defaultJSP>
    </regions>
</taskDisplay>
```

**Payload File for the Autogenerated JSP**

Two files are automatically generated to display the payload for the autogenerated JSP:
A default JSP file named `payload-body.jsp`. This file is added to the HTML root directory of your project, which is by default the `public_html` directory.

A mapping XML file named `payload-body.xml`. This file is added to the same directory of your project as `payload-body.jsp`.

**Note:** If you select Custom Task Form in Step 2 on page 15-67, you can specify a unique file name for the autogenerated JSP. The mapping XML file is created based on the JSP file name. You can also select the payload elements to include in the autogenerated JSP. For example, if the JSP file is named `autogenerate-body.jsp`, then the mapping XML file is named `autogenerate-body.xml`. See "Generating a Custom Task Display Form" on page 15-74 for details.

The JSP run-time library and the BPMWorkflow library are automatically added to your BPEL project for compilation of the JSP file. The default JSP is designed with two goals in mind:

- To present you with a simple form, that is, an XSD tree with a depth of more than three must be shown in a more readable way in the JSP.
- The default JSP must require minimum modification. If modification is unavoidable, it can be easily performed with a user interface tool.

To attain these goals, instead of presenting a tree structure that mimics the payload XSD structure, the default JSP groups the entire payload structure in sections. It groups simple types that belong to the same parents and makes them sections.

For example, assume you provide the following payload XSD:

```xml
<schema xmlns="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.mycompany.com/mycompany"
    xmlns:mp="http://www.mycompany.com/mycompany">  
  <element name="myCompany" type="mp:myCompanyType"/>
  <complexType name="myCompanyType">
    <sequence>
      <element name="board" type="mp:boardType"/>
      <element name="CEO" type="string"/>
      <element name="department" type="mp:departmentType" maxOccurs="unbounded"/>
    </sequence>
  </complexType>
  <complexType name="boardType">
    <sequence>
      <element name="size" type="int"/>
      <element name="head" type="string"/>
    </sequence>
  </complexType>
  <complexType name="departmentType">
    <sequence>
      <element name="size" type="int"/>
      <element name="head" type="string"/>
      <element name="function" type="string"/>
    </sequence>
  </complexType>
</schema>
```

Note: If you select Custom Task Form in Step 2 on page 15-67, you can specify a unique file name for the autogenerated JSP. The mapping XML file is created based on the JSP file name. You can also select the payload elements to include in the autogenerated JSP. For example, if the JSP file is named `autogenerate-body.jsp`, then the mapping XML file is named `autogenerate-body.xml`. See "Generating a Custom Task Display Form" on page 15-74 for details.
This XSD has the structure shown in Figure 15–33.

The section department is different from other sections and can have multiple occurrences (maxOccurs > 1). In this case, all the fields in this section (that is, size, head, and function) are horizontally presented in a table, with each row representing one department. This is called a table section. There is a plus (+) button for adding a row (department) and a minus (-) button for subtracting a row (department) for the department table section.

Unlike a regular section, it is not necessarily true that all the fields belong to the same XSD parent in a table section. For example, suppose the head element has two elements: employeeNumber and dateOfBirth. Since these two elements have maxOccurs set to less than or equal to 1, they are shown in the same department table section. This is a desired behavior, because adding a row in the department table not only adds a size and a function field, but also adds the head information fields in the same department row. This makes it easy to move through complicated payload instances.

Nested multiple (maxOccurs > 1) elements are supported. Assume the department element has a groupMember child element whose maxOccurs is unbounded. In that case, the parent element department is presented in a table section while the child groupMember elements are presented in different child table sections. The parent department table section has a column called group member that contains an HTML href link pointing to its corresponding child group member section in each department row. Pressing the + button in the parent department section not only adds a row in the parent table, but also adds a child section for that corresponding new row.
The default JSP in the current release has the following limitations:

- XSDs that contain recursive elements are not supported.
- The default JSP shows all the simple types defined in the payload XSD. If multiple simple types belong to the same XSD choice block, all these simple types are shown in the default JSP. Although simple types are preserved in the JSP, XSD restrictions are not relevant.
- Only payloads copied from variables that are not simple types are supported. For example, if the query is `bpws:getVariableData(var)` or `bpws:getVariableData(var, part)` and the variable is a simple type, then JSP generation fails. Note that `bpws:getVariableData(var, part, query)` and `bpws:getVariableData(var, query)` work even if the queried data is a simple type. You only need to make sure the variable itself is not a simple type.
- XSI extensions are not supported
- No special handling of XSD order indicators occurs (that is, choice, all, and sequence). For example, the default JSP does not check if you entered both firstname and lastname:

```xml
<xs:element name="person">
  <xs:complexType>
    <xs:all>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
    </xs:all>
  </xs:complexType>
</xs:element>
```

Customizing the Autogenerated JSP The autogenerated default JSP is generic, and so may require changes to improve its look and feel. The JSP works in conjunction with the mapping file to determine which elements in the payload are displayed in the form.

Customizing the Mapping File The mapping file gives you control of the presentation. The mapping file is an XML file that contains a list of viewable fields. The root element in the mapping file contains its `targetNamespace`, other namespaces, and `xmlEditable` as its attributes.

All the elements that are simple types are listed as fields in the mapping file. Along with these elements, their immediate parents are listed as well for multilanguage support. Each field has three properties defined in the mapping file. They are `xpath`, `editable`, and `resource_key`.

The `xpath` property defines the XPath of this field. It is always prefixed by `/ns0:task/ns0:payload`. This is the XPath to the root of the payload object. When `maxOccurs` is greater than 1, it is denoted by `[*]`. For example, `/ns0:task/ns0:payload/company[*]/ceo` shows that `maxOccurs` is greater than 1 for the company field.

**Note:** Do not modify this XPath field because it is also a unique key that determines the identity of the field.

The `editable` property defines if this field is editable. It defaults to `true`. If the value of this field is changed to `false`, the default JSP shows a disabled text field that disallows value changes.
The `resource_key` property is for multilanguage support. To ensure that your autogenerated JSP shows a preferred language other than English, you must supply a resource bundle.

Follow these steps to add a resource bundle:

1. Create a bundle file (for example, `MyBundle`). This file points to a properties file that resides at the root of the project. The following code shows an example of `MyBundle_en-US.properties`:

```
ACCEPT_MSG = Accept
REJECT_MSG = Reject
FLEX_STRING1_MSG = Flex String1
FLEX_LONG1_MSG = Flex Long1
FLEX_DATE1_MSG = Flex Date1
TASK_TITLE = i18n Task
```

In this case, if a field is defined in your mapping file as follows

```xml
<field>
  <xpath>/ns0:task/ns0:payload/taskTitle</xpath>
  <editable>true</editable>
  <resourceKey>TASK_TITLE</resourceKey>
</field>
```

then calling

```
FormUtil.getElementDisplayName("/ns0:task/ns0:payload/taskTitle", form, context.getLocale(), task)
```

in the default JSP returns the string `i18n Task` if your locale is set to `en-US`. Similarly, if your locale is set to French, the proper properties file (`MyBundle_fr.properties`) is picked up.


Customizing the Default JSP

If the mapping file does not provide enough control, you can modify the default JSP file. Only modify the section after the label:

```
/* Modify the code below when necessary */
```

Most JSP modifications can be made in the JSP design view of Oracle JDeveloper.

By default, all the fields are set to text field. If you want to change a text field to a text area, you can do the following:

1. Select Text Area in the Component Palette, as shown in Figure 15–34.
2. Drop it into the position of the text field you want to replace.

3. Note that the name of the text field is set by calling the function
   `oracle.bpel.services.workflow.worklist.payload.PayloadFormGenerator.constructName(String xpath)`, and the value of the field is set by `PayloadFormGenerator.selectNodeValue(Element payload, String xpath, Map namespace)`. These functions must be used to construct form field names and to retrieve form field values.

4. Set the text area’s name and value to the same name and value as the text field.

5. Delete the text field.

6. In the place you want to insert text or other HTML elements that are not in a table, add text by typing it or add an HTML element by dragging and dropping the HTML component from the Component Pallet.

7. If the place you want to insert HTML elements is in an HTML table, to insert text or a horizontal rule, first add a table row by clicking a row, right-clicking, and selecting Insert Row. After a row is inserted, you may need to merge all the cells in the row by selecting all the cells in the row and right-clicking to select Merge Cells. Then you can either type your text or drag and drop your HTML component.

8. If you want to change the layout of the table or form, highlight the section you want to modify, right-click, and select table or form.

9. If you want to format the text, use the toolbar’s color and style buttons.

   It is recommended that you modify the default JSP’s look and feel only. You should preserve the functions being used in the JSP. You must not alter the hidden parameters being submitted in the HTML form, because the Update button invokes form submission to the WFTaskUpdate that expects certain values. If
your change is complicated and has programming logic in it, you must switch to
the source view and modify the JSP code directly.
By putting the statement \(<% page pageEncoding="UTF-8" %>\) in the default
JSP, UTF-8 is set as the default encoding.

**See Also:** The HelpDeskServiceRequest demo in SOA_Oracle_
Home\bpel\samples\demos for an example of an autogenerated
JSP and how to change the payload presentation

### Displaying a Check Box on the Worklist Payload JSP

Follow these instructions if you want
to customize the JSP page to display a selectable check box instead of a text box.

1. Note that the input for the status is generated as follows.

   ```html
   <input
   name="\(<%=PayloadFormGenerator.constructName("/ns0:task/ns0:payload/ns1:holds/ns1:holdCodes[" + i3 + "]/ns1:status")%>"
   value="\(<%=PayloadFormGenerator.selectNodeValue(payload, "/ns0:task/ns0:payload/ns1:holds/ns1:holdCodes[" + i3 + "]/ns1:status", form.getNameSpaceMap())%>"
   dataType="string" <%=thisDisabled%> />
   ```

2. Substitute the entire code block shown in Step 1 with the following code block:

   ```java
   String checked = "";
   String status = PayloadFormGenerator.selectNodeValue(payload, "/ns0:task/ns0:payload/ns1:holds/ns1:holdCodes[" + i3 + "]/ns1:status", form.getNameSpaceMap());
   if(status != null && status.equals("true") { // checked = "checked";
   }
   ```

   ```html
   <input type="CHECKBOX"
   name="\(<%=PayloadFormGenerator.constructName("/ns0:task/ns0:payload/ns1:holds/ns1:holdCodes[" + i3 + "]/ns1:status")%>" value="\(<%=status%>" <%=checked%>
   onClick="changeStatusValue(this)" />
   ```

3. Add the following JavaScript. This is required because the value for the check
box field in JavaScript is always the value defined in the input element.

   ```javascript
   function changeStatusValue(obj)
   {
   obj.value = obj.checked;
   }
   ```

### Generating a Custom Task Display Form

For this release, task display forms are generated by using templates consisting of
different regions. Oracle JDeveloper automatically includes three templates and two
default JSPs:

The three templates are as follows:

- **Three Region JSP** — Consists of the header, body and footer regions. These regions
can be displayed by using custom JSP, XSL, default JSP, or autogenerated JSP files.
The automatically generated JSP displays the body region.

- **Two Region JSP** — Consists of the header and footer regions

---

15-74  Oracle BPEL Process Manager Developer's Guide
Task 3: Generating the Task Display Form

- One Region JSP — Consists of the body region
  The two default JSPs are as follows:
- The header JSP displays task attributes such as task number, priority, title, and so on.
- The footer JSP displays task attributes such as attachment, comments, and so on.

The custom task display form enables you to select the template and rendering type for displaying task details.

1. Select Custom Task Form from the list shown in Step 2 on page 15-67.
   The Task Form Display window appears.

2. Select a template from the Current Template list. Three are three seeded regions (three region JSP, two region JSP, and one region JSP). After selecting a region, you can specify how to render it.

3. See the following sections for details about generating the different types of custom task display forms:

<table>
<thead>
<tr>
<th>Type</th>
<th>See...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Auto JSP</td>
<td>&quot;Autogenerated JSP&quot; on page 15-75</td>
</tr>
<tr>
<td>Custom JSP</td>
<td>&quot;Custom JSP&quot; on page 15-76</td>
</tr>
<tr>
<td>Default JSP</td>
<td>&quot;Default JSP&quot; on page 15-77</td>
</tr>
<tr>
<td>XSL</td>
<td>&quot;XSL&quot; on page 15-77</td>
</tr>
</tbody>
</table>

**Autogenerated JSP**

This option enables you to automatically generate a form for the payload of the task. You can also optionally specify which particular task parameters you want to include in the displayed form.

1. Select Auto JSP from the Body list in the Rendering section.
   An icon displays to the right of Body.jsp in the Source section.
2. Click the icon.
   The Payload Mapping window appears.
3. Select message attributes to include in the autogenerated JSP.
4. Click OK to return to the Task Form Display window.

**Custom JSP**

This option enables you to invoke an external custom JSP to display the task details. You can also specify URL parameters to pass to this JSP at run time. Three parameters are passed in by default — taskID, version, and workflowContext. Additional parameters must be explicitly specified.

1. Select **Custom JSP** from the **Header** list in the **Rendering** section.
   
   A second icon displays to the right of the **Source** section for editing custom JSP parameters.

2. Enter the custom JSP file name in the **Source** field or click the first icon to select the JSP file to use. This JSP is used in the project and deployed with the other JSP files.

3. Click the second icon to specify run time JSP parameters.
   
   The Payload Mapping window appears. This window enables you to add input JSP parameters.
4. Add a parameter by clicking the + sign.
5. Add a name in the Name column.
6. Click the icon to the right of the row to display the Expression Builder window to dynamically enter a value for the XPath column.

For this example, the custom JSP is using a parameter named PRIORITY to receive the task ID from the request. Therefore, PRIORITY is specified as the name and /tns:task/tns:systemAttributes/tns:PRIORITY is specified as the XPath expression.

See Also: "Creating Custom JSP Forms" on page 15-78 for details about explicitly passing parameters

Default JSP
This option provides the default Header1.jsp and Footer1.jsp files to display the header and footer regions, respectively.

XSL
This option enables you to specify an XSL to convert the task XML document into an HTML document for the form. Note that this is useful only to create read-only forms.
1. Enter the HTTP location in the Source field or click the first icon to select the input XSL file to use.

Deploying Task Display Forms
Workflow task display forms are deployed by using the deployTaskForm ant target. This target is executed when you deploy the BPEL process from Oracle JDeveloper or from the command prompt. This target generates an EAR file that includes all generated default or custom JSPs. This generated EAR file is deployed as a child of the Oracle BPEL Process Manager application.
The following directory structure is generated.

```
JDev_Oracle_Home\dev\mywork\application_name\project_name\public_html\human_task_name\form
```

The following subdirectories and files are created:

- A J2EE enterprise archive directory named *ear* is created. EAR deployment descriptors are generated and stored in the *META-INF* subdirectory.
- A Web archive (WAR) directory named *war* is created. This directory contains the following files and subdirectories:
  - Style sheets and Java server page files for the header (*Header1.jsp*), footer (*Footer1.jsp*), and body (*payload-body.jsp* and *payload-body.xml*) are generated and stored in the *war* directory.
  - Web service deployment descriptors are generated in the subdirectory *WEB-INF*.

You can delete all form-related files by right-clicking the human task folder in the Application Navigator and selecting Delete Task Form files.

Creating Custom JSP Forms

As described earlier, you can register a custom JSP for rendering the task details in the worklist. The BPEL worklist invokes any custom JSP that has been registered.

Follow these instructions to create a custom JSP form.

1. Get the task ID, version, and context ID from the request.
2. Get the workflow context object based on the context ID.
3. Get the task object based on the task ID and version. Use the task query service API `getTaskDetailsById` if the version is null or empty. Otherwise, use the `getTaskVersionDetails` API.
4. Use the task object methods to get the values you want to display in the JSP.
5. In the case of update support, generate the hidden HTML type for the following parameters, so that the update servlet can read these parameter values:

```java
oracle.bpel.service.workflow.worklist.api.payload.PayloadConstants.WORKLIST_NEXT_PAGE_PARAMETER_NAME
oracle.bpel.service.workflow.worklist.api.payload.PayloadConstants.WORKLIST_LOGIN_PAGE_PARAMETER_NAME
oracle.bpel.service.workflow.worklist.api.payload.PayloadConstants.WORKLIST_ERROR_PAGE_PARAMETER_NAME
```

You can get the values for these parameters in the custom JSP servlet request object. Run time invokes the custom JSP by passing these parameters.

The following custom JSP code shows how to use these steps to write a custom JSP that uses the local query service and verification APIs. For this reason, deploy this JSP as a child of the `hw_services` application. If you do not want to deploy to the same application server, replace local APIs with remote APIs.

```jsp
<%@ page contentType="text/html;charset=UTF-8"%>
<%@ page import="java.util.*,java.net.URLEncoder,java.io.UnsupportedEncodingException,java.text.*",
```
Task 3: Generating the Task Display Form

```java
String taskId = request.getParameter(Constants.WORKLIST_TASKID_PARAMETER_NAME);
String strTaskVersion = request.getParameter(Constants.WORKLIST_TASK_VERSION_PARAMETER_NAME);
String contextId = request.getParameter(Constants.WORKLIST_CONTEXT_PARAMETER_NAME);

// no need to use Notm to get the task
Task task = (Task)session.getAttribute(Constants.SESSION_CURRENT_TASK_OBJECT);
IVerificationService verificationService = VerificationService.getVerificationService();
IWorkflowContext context = verificationService.getContext(contextId);

if(task == null) {
    ITaskQueryService queryService = TaskQueryService.getInstance();
    if(taskVersion == 0) {
        task = queryService.getTaskDetailsById(context, taskId);
    } else {
        // TO DO
    }
}

// no need to use Notm to get the task
Task task = (Task)session.getAttribute(Constants.SESSION_CURRENT_TASK_OBJECT);
IVerificationService verificationService = VerificationService.getVerificationService();
IWorkflowContext context = verificationService.getContext(contextId);

if(task == null) {
    ITaskQueryService queryService = TaskQueryService.getInstance();
    if(taskVersion == 0) {
        task = queryService.getTaskDetailsById(context, taskId);
    } else {
        // TO DO
    }
}
```

Oracle BPEL Process Manager Workflow Services 15-79
task = queryService.getTaskVersionDetails(context, taskId, taskVersion);
} }
Locale locale = context.getLocale();
// get the TaskId and use above object
SimpleDateFormat dfshort = new SimpleDateFormat( "MM/dd/yy" );
SimpleDateFormat dflong = new SimpleDateFormat("MM/dd/yy hh:mm a");
String nextPage = request.getParameter(Constants.WORKLIST_NEXT_PAGE_          PARAMETER_NAME);
String loginPage = request.getParameter(Constants.WORKLIST_LOGIN_PAGE_     PARAMETER_NAME);

Adding Update Support in the Custom JSP
To add update support in the custom JSP, you can write the servlet that uses the remote task service APIs to update the custom JSP task values:
1. Get the task object by using the same steps as used in the custom JSP.
2. Query the task object and set the values based on the custom JSP form. For example, if the custom JSP form allows a user to update the priority attribute, then get the priority JSP form value and call task.setPriority(newValue);
3. Use the remote task service API to update the task.
4. Get the value from servlet parameter WORKLIST_NEXT_PAGE_PARAMETER_NAME, which the custom JSP page includes as a hidden parameter.
5. Redirect the page to the URL.

How Changes to a Workflow Appear in Worklist Application
Changes made in Oracle BPEL Control to a BPEL process that includes a human task impact how tasks display in Oracle BPEL Worklist Application:
- If you abort an active BPEL process instance on the Instances tab, associated tasks are marked as Stale in the Status column of the Oracle BPEL Worklist Application home page.
- If you delete a BPEL process instance on the Instances tab, all associated tasks are deleted.
- If you undeploy a BPEL process on the BPEL Process tab, associated tasks are marked as Stale in the Status column of the Oracle BPEL Worklist Application home page.

Notifications from Workflow Services
Notifications are sent to alert users of changes to the state of a task. Notifications can be sent through any of the following channels: e-mail, telephone voice message, fax, pager, or SMS.
This section contains the following topics:
- Configuring the Notification Channel
- Contents of Notification
Configuring Messages in Different Languages
Sending Actionable E-mails
Sending Inbound and Outbound Attachments
Sending Inbound Comments
Reliability Support
Sending Secure Notifications
Channels Used for Notifications
Sending Reminders

Configuring the Notification Channel

After configuring the notification service for e-mail and other channels in Oracle JDeveloper, set the NotificationMode parameter for the notification service to either ALL or EMAIL in the SOA_Oracle_Home\bpe\system\services\config\ns_emails.xml file.

By default, this value is set to NONE, meaning that no notifications are sent. The possible values for the NotificationMode attribute are:

- **ALL** – the e-mail, SMS, voice, fax, and pager channels are configured and notification is sent through any channel.
- **EMAIL** – Only the e-mail channel is configured for sending notification messages.
- **NONE** – No channel is configured for sending notification messages. This is the default setting.

The notifications for a task can be configured during the creation of a task in the Human Task editor. Notifications can be sent to different types of participants for different actions. The actions for which a task notification can be sent are as follows:

- **Assigned** — when the task is assigned to users or a group. This action captures the following task actions — acquire, adhoc route, delegate, escalate, information for a task is submitted, push back, reassign, release, and resume.
- **Task is completed**
- **Task is errored**
- **Task is expired**
- **Information is requested for a task**
- **Task outcome is updated**
- **Task is suspended**
- **Task is withdrawn**

Notifications can be sent to users involved in the task in various capacities. This includes:

- **Assignees** – the users or groups to whom the task is currently assigned
- **Initiator** – the user who created the task
- **Creator** – the user who created the task
- **Approvers** – the users who have approved the task so far
Notifications from Workflow Services

This applies to a sequential list of approvers participant type where multiple users have approved the task and a notification must be sent to all.

- Owner – the owner of the task

When the task is assigned to a group, each user in the group is sent a notification if no notification endpoint is available for the group.

See Also:
- "Specifying Participant Notification Preferences" on page 15-43 to configure task notifications in the Human Task editor
- Chapter 14, "Oracle BPEL Process Manager Notification Service"
- Service Configuration chapter of the Oracle BPEL Process Manager Administrator's Guide for details about editing the ns._emails.xml file and (for the JAZN XML provider) users-properties.xml file

Contents of Notification

Each e-mail notification can contain the following parts:

- The notification message
- The HTML content from the worklist application — This is a read-only view of the worklist application on the task.
- Task attachments — If the notification includes task attachments
- Actionable links

Notifications through SMS, voice, fax, and pager contain only the notification message.

The notification message is an XPath expression that can contain static text and dynamic values. In creating the messages, only the task BPEL variable is available for dynamic values. This restriction is because the messages are evaluated outside the context of the BPEL process. The payload in the task variable is also strongly typed to contain the type of the payload for XPath tree browsing. The XPath extension function hwf:getNotificationProperty(propertyName) is available to get properties for a particular notification. The function evaluates to corresponding values for each notification. The propertyName can one of the following values:

- recipient — The recipient of the notification.
- recipientDisplay — The display name of the recipient.
- taskAssignees — The task assignees.
- taskAssigneesDisplay — The display names of the task assignees.
- locale — The locale of the recipient.
- taskId — The ID of the task for which the notification is meant.
- taskNumber — The number of the task for which the notification is meant.
- appLink — The HTML link to the worklist application task details page.

The following example demonstrates the use of hwf:getNotificationProperty and hwf:getTaskResourceBundle together:

```
concat('Dear ', hwf:getNotificationProperty('recipientDisplay'), ',
	Task ',
	/task/task/task:systemAttributes/task:taskId,' is assigned to you.
	')
```
Notifications from Workflow Services

```
hwf:getTaskResourceBundleString(/task:task/task:systemAttributes/task:taskId,
    'CONGRATULATIONS', hwf:getNotificationProperty('locale'))
```

This results in a message similar to the following:

Dear Cooper, James Task 1111 is assigned to you. Congratulations

Configuring Messages in Different Languages

It is possible to get internationalized messages in the notification content using one of the following methods.

- If you want to use values from the resource bundle specified during the task definition, use the XPath extension function `hwf:getTaskResourceBundleString(taskId, key, locale?)`. This function returns the internationalized string from the resource bundle specified in the task definition.
  - The locale of the notification recipient can be retrieved with the function `hwf:getNotificationProperty('locale')`.
  - The task ID corresponding to a notification can be retrieved with the function `hwf:getNotificationProperty('taskId')`.
- If a different resource bundle is used, the XPath extension function `orc1:get-localized-string()` can be used to retrieve localized messages.

See Also: "Specifying Multilingual Settings" on page 15-48

Sending Actionable E-mails

Task actions can be performed through e-mail if the task is set up to enable actionable e-mail (the same actions can also be performed from the Oracle BPEL Worklist Application). An actionable e-mail account is the account in which task action-related e-mails are received and processed. This e-mail account name is identified by the element `actionableEmailAccountName` in the configuration file `SOA_Oracle_Home\bpel\system\services\config\wf_config.xml`.

Ensure that you select Make e-mail messages actionable in the Notification Settings section of the Human Task editor to make e-mail notifications actionable. (See Figure 15-25 on page 15-44.) This enables you to perform task actions through e-mail.

If a notification is actionable, the e-mail contains links for each of the custom outcomes. Clicking on the links invokes the compose window of the e-mail client. You do not have to change anything in the subject or the body in this e-mail. If you change the content with the NID substrings, the e-mail is not processed.

Figure 15-35 shows an actionable e-mail sample:
Sending Inbound and Outbound Attachments

If the include attachments flag is checked, only e-mail is sent. The e-mails include all the task attachments as e-mail attachments. Select Send task attachments with e-mail notifications in the Notification Settings section of the Human Task editor. (See Figure 15–25 on page 15-44.)

In the actionable e-mail reply, the user can add attachments in the e-mail and these attachments are added as task attachments.

Sending Inbound Comments

In the actionable e-mail reply, the user can add comments in the e-mail between Comments[['' and '']] and those contents are added as task comments. For example, Comments[['looks good']].

Reliability Support

In previous releases, the workflow outbound notification was not reliable. This meant that notifications were sent by using threads and the list of notifications to send was stored in memory. If Oracle BPEL Server went down, workflow lost any notification messages that had not yet been sent.

With release 10.1.3, the workflow outbound notification service uses queues with the persistency service to send notifications to users.
Whenever a workflow needs to send a notification to a user, it stores the task information such as notification ID, task ID, version, and so on in the dehydration store and enqueues the notification ID to the queue. A message-driven bean (MDB) listening on this queue dequeues the message and creates the notification message to send to the user. It then uses the notification service to send this message, which uses the queue with the dehydration store.

**See Also:** Chapter 14, "Oracle BPEL Process Manager Notification Service" for additional details about the reliable notification service

### Sending Secure Notifications

If a notification is marked as secure in the Notification Settings section of the Human Task editor, a default notification message is used. (See Figure 15–25 on page 15-44.) The default notification message includes a link to the task in the Oracle BPEL Worklist Application. You must log in to see task details.

**See Also:** "Securing Notifications, Making Messages Actionable, and Sending Attachments" on page 15-46

### Channels Used for Notifications

The channel through which a user is notified is determined by the notification preference attribute of the user specified in JAZN. The notification preference is identified by the attribute `orclWorkflowNotificationPreference`. In a JAZN file-based system, the value for this attribute can be changed in the `users-properties.xml` file located at `SOA_Oracle_ Home\bpel\system\services\config`.

In an Oracle Internet Directory-based system, the user properties can be changed using the Oracle Delegated Administration Service. If this attribute is not set, the e-mail channel is used as the default.

**See Also:** Oracle Identity Management Guide to Delegated Administration for more information on the Oracle Delegated Administration Service

### Sending Reminders

Tasks can be configured to send reminders, which can be based on the time the task was assigned to a user or the expiration time of a task. The number of reminders and the interval between the reminders can also be configured. The message used for reminders is the message that is meant for ASSIGNEES when the task is marked as ASSIGNED.

You set reminders in the Notification Settings section of the Human Task editor. (See Figure 15–25 on page 15-44.) Reminder configuration involves these parameters.

- **Recurrence** — The recurrence specifies the number of times reminders are sent. The possible values for recurrence are EVERY, NEVER, 0, 1, 2 ..., 10.

- **RelativeDate** — The `relativeDate` specifies if the reminder duration is computed relative to the assigned date or to the expiration date of the task. The possible values for the `relativeDate` are ASSIGNED and EXPIRATION.

- **Duration** — The duration from the `relativeDate` and the first reminder and each reminder since then. The data type of duration is `xsd:duration`, whose format is defined by ISO 8601 under the form `PnYnMnDTnHnMnS`. The capital
letters are delimiters and can be omitted when the corresponding member is not used. Examples include PT1004199059S, PT132S, PT2M105, P1DT2S, -P1Y, or P1Y2M3DT5HS2OM30.123E.

The following examples illustrate when reminders are sent.

- If the `relativeDate` is `ASSIGNED`, the `recurrence` is `EVERY`, and the reminder duration is `PT1D`. If the task is assigned at 3/24/2005 10:00 AM, then reminders are sent at 3/25/2005 10:00 AM, 3/26/2005 10:00 AM, and so on until the user acts on the task.
- If the `relativeDate` is `EXPIRATION`, the `recurrence` is 2, the reminder duration is `PT1D`, and the task expires at 3/26/2005 10:00 AM, then reminders are sent at 3/24/2005 10:00 AM and 3/25/2005 10:00 AM if the task was assigned before 3/24/2005 10:00 AM.
- If the `relativeDate` is `EXPIRATION`, the `recurrence` is 2, the reminder duration is `PT1D`, the task expires at 3/26/2005 10:00 AM, and the task was assigned at 3/24/2005 3:00 PM, then only one reminder is sent at 3/25/2005 10:00 AM.

See Also: “Setting Up Reminders” on page 15-46

### End-to-End Workflow Examples

Table 15–14 shows the end-to-end workflow examples included with Oracle BPEL Process Manager. Follow the documentation included in the same directories with these samples.

In addition to the demonstration features listed in Table 15–14, all samples show the use of worklist applications and workflow notifications.

<table>
<thead>
<tr>
<th>Sample</th>
<th>Location</th>
<th>Description</th>
<th>Demonstrates</th>
</tr>
</thead>
<tbody>
<tr>
<td>AutoLoanDemo</td>
<td>SOA_Oracle_Home/bpel/samples/demos</td>
<td>Review and approve a loan request</td>
<td>Single approval</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Integration with a business rule engine</td>
</tr>
<tr>
<td>DocumentReview</td>
<td>SOA_Oracle_Home/bpel/samples/demos</td>
<td>Review and approve a document</td>
<td>Group vote</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Adding attachments to tasks</td>
</tr>
<tr>
<td>ExpenseRequestA</td>
<td>SOA_Oracle_Home/bpel/samples/demos</td>
<td>The ExpenseRequest business process is used to approve and reject an expense request from an employee</td>
<td>Management chain approval</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Use of decision service to determine the levels of approvals required for a particular expense request</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Microsoft Office integration</td>
</tr>
<tr>
<td>HelpDeskService Request</td>
<td>SOA_Oracle_Home/bpel/samples/demos</td>
<td>Approval of a help desk service</td>
<td>Adhoc approval</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Custom worklist user interface</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Promotion of task payload message attributes</td>
</tr>
</tbody>
</table>
Vacation Request Example

This example describes how to create a vacation request business process. In this business process, the manager of a user requesting a vacation approves or rejects the request. The approval or rejection is a one-step process.

This example highlights the use of the following:

- Modeling a single approval workflow using Oracle JDeveloper
- Using the Oracle BPEL Worklist Application to view and respond to tasks

Prerequisites

This example assumes the following:

- You are familiar with basic BPEL constructs, including BPEL activities and partner links, and basic XPath functions. Familiarity with Oracle JDeveloper—the environment for creating and deploying BPEL processes—is also assumed.
- You must configure the e-mail server settings for the account Default to enable e-mail notifications. The Default account is used to send e-mails. The e-mail server configuration is in:

  `<SOA_Oracle_Home>/bpel/system/services/config/nas_emails.xml`

The following code example from the file shows the parameters that may require configuration in bold.

```xml
<EmailAccount>
  <Name>Default</Name>
  <GeneralSettings>
    <FromName>Oracle BPM</FromName>
    <FromAddress>account2@yourdomain.com</FromAddress>
  </GeneralSettings>
  <OutgoingServerSettings>
    <SMTPHost>yourdomain.com</SMTPHost>
    <SMTPPort>25</SMTPPort>
  </OutgoingServerSettings>
  <IncomingServerSettings>
</EmailAccount>
```

<table>
<thead>
<tr>
<th>Sample</th>
<th>Location</th>
<th>Description</th>
<th>Demonstrates</th>
</tr>
</thead>
<tbody>
<tr>
<td>LoanDemoPlus</td>
<td>SOA_Oracle_Home/bpel/sample</td>
<td>Approval of a loan application</td>
<td>Group assignment (in StarLoan process)</td>
</tr>
<tr>
<td></td>
<td>$demos</td>
<td></td>
<td>Custom worklist user interface (in LoanFlowPlusUI and StarLoanUI)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>FYI tasks (in LoanFlowPlus process)</td>
</tr>
<tr>
<td>OrderApproval</td>
<td>SOA_Oracle_Home/bpel/sample</td>
<td>Approve or reject a purchase order</td>
<td>Sequential workflow</td>
</tr>
<tr>
<td></td>
<td>$demos\tutorial\127, OrderBookingTutorial</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VacationRequest</td>
<td>SOA_Oracle_Home/bpel/sample</td>
<td>Vacation request approval or denial</td>
<td>Simple workflow</td>
</tr>
<tr>
<td></td>
<td>$demos</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
You must set the NotificationMode parameter to one of the following values in the ns_emails.xml file:

- **ALL** - If you have the e-mail, SMS, voice, fax, and pager channels set up.
- **EMAIL** - If you have only the e-mail channel set up.

For example:

```xml
<EmailAccounts xmlns="http://xmlns.oracle.com/ias/pbipel/NotificationService"
    EmailMimeCharset="" NotificationMode="EMAIL">
```

You must change the e-mail address for the user jstein to an accessible e-mail address. If the XML-based JAZN provider is used, these properties can be changed in:

```
SOA_Oracl_Home\bpel\system\services\config\users-properties.xml
```

The following XML segment from the users-properties.xml shows where the e-mail is configured:

```xml
<userObject>
    <name>jstein</name>
    <description>Demo User</description>
    <email>user1@dlsun4254.us.oracle.com</email>
    ...
    <notificationPreferences>Mail</notificationPreferences>
</userObject>
```

You must restart Oracle BPEL Process Manager after making any of the preceding changes.

**Modeling the Vacation Request Process**

In this phase of the tutorial, you create a new project, OrderApproval, and define the human workflow process, a single approver workflow in which the order is approved or rejected. The order is first assigned to the Supervisor role. After a user with the Supervisor role approves the order, it is sent to the user’s manager for final approval.

This section contains these tasks:
End-to-End Workflow Examples

- Creating the Vacation Request Process and Importing the Schema
- Adding a Human Task to the Order Approval Process
- Assigning Input and Output Parameters for the Human Task
- Creating a Task Form for the Worklist
- Modeling the Task Outcome
- Validating, Compiling, and Deploying the Order Approval Process
- Running the Order Approval Process

Creating the Vacation Request Process and Importing the Schema
1. Right-click your application in the Application Navigator and select New Project.
2. Select BPEL Process Project.
3. Create an asynchronous BPEL process with the name VacationRequest.
4. Click Next.
5. Click the flashlight next to Input Schema Element to browse for VacationRequest.xsd in
   `SOA_Oracle_Home\bpel\samples\demos\VacationRequest\bpel`
6. Click Open.
7. The Type Chooser window appears.
8. Expand and select Imported Schemas > VacationRequest.xsd > VacationRequestProcessRequest.
9. Click OK.
10. Click the flashlight next to Output Schema Element.
12. Click Finish.

The schemas are now imported into the project. VacationRequest.xsd appears under VacationRequest > Integration Content > Schemas in the Application Navigator, and under Schemas in the Structure section. The BPEL process—a Receive activity (receiveInput) and an Invoke activity (callbackClient)—is displayed.

Adding a Human Task to the Order Approval Process

Summary: When you define the human task, the VacationApproval.task file—the task configuration metadata file—is created.

1. Drag and drop a Human Task activity between receiveInput and callbackClient.
2. Click the Create Task Definition icon (second icon).
3. Enter VacationApproval for the human task name and click OK. (Accept the default location.)
   The VacationApproval.task file is created.
   The Human Task editor is displayed.
4. For Title, enter Vacation Approval.
5. Accept the default values for Priority and Outcomes.
6. For Parameters, click the + icon on the right side of the window.
   The Add Task Parameter window is displayed.
7. Click Element and then the flashlight icon.
8. In the Type Chooser window, expand and select Project Schema Files > VacationRequest.xsd > VacationRequestProcessRequest, and click OK.
9. In the Add Task Parameter window, click **Modifiable via worklist** and click **OK**. This ensures that you can modify task data using the Oracle BPEL Worklist Application.

10. In the **Assignment and Routing Policy** section, click the + icon on the right side of the window.
    The Add Participant Type window is displayed.

11. For **Type**, select **Single Approver**.
    This participant type acts alone on the task.

12. For **Label**, enter **Vacation Approver**.
    This participant type acts alone on the task.

13. Click **By expression**.
    In this example, you assign the task to the manager of the vacation requester.

14. Click the icon to the right of the **Dynamic User Xpath** field to display the Expression Builder window.

15. Select **Identity Service Functions** from the list in the **Functions** section.

16. Double-click **getManager**.

17. Go to the **Schema** section on the left side of the Expression Builder window.

18. Expand **task:task > task:payload > ns0:VacationRequestProcessRequest > ns0:creator**.

19. Click **Insert Into Expression**.
    The Expression Builder window appears as follows:
20. Click OK to return to the Add Participant Type window.
21. Click OK to return to the Human Task editor.
22. Click the + sign to expand the Expiration and Escalation Policy section.
23. Select Expire after from the drop-down list.
24. Click Fixed Duration and select 1 from the Day list.
25. Select Save from the File main menu.
26. Click the X next to VacationApproval.task to close the Human Task editor.

Assigning Input and Output Parameters for the Human Task

**Summary:** Map the fields to the variables in the BPEL process.

1. Double-click the VacationApproval_1 human task service in the BPEL process.

This displays the Human Task window.

2. In the Task Title field, enter the word for after the words Vacation Approval.
3. Click the icon at the right to display the Expression Builder window.
4. In the BPEL Variables section, expand and select inputVariable > payload > client:VacationRequestProcessRequest > client:creator.
5. Click Insert Into Expression.
The XPath expression appears in the Expression section.

6. Click OK.

The XPath expression is appended to the task title.

7. Click the icon to the right of the Initiator field to display the Expression Builder window.

8. Repeat Steps 4 through 6 to insert the same XPath expression in the Initiator field.

9. Click the flashlight icon under the BPEL Variable column.

The Task Parameters window appears.

10. In the Task Parameters window, expand and select Variables > inputVariable > payload > client:VacationRequestProcessRequest.

11. Click OK.

12. In the Human Task window, click OK.
Creating a Task Form for the Worklist

Summary: An autogenerated task form, payload-body.jsp, is created.

1. In the Application Navigator, right-click the VacationApproval folder and select Auto Generate Simple Task Form.
   This automatically generates a task form file.
2. Close payload-body.jsp by clicking the X sign at the top.

Modeling the Task Outcome

Summary: The Switch activity reflects the possible outcomes, or cases, specified previously, Approve and Reject. It also has an Otherwise case to represent other outcomes, such as errored, stale, or expired. Inside each of the cases, you can add activities to complete modeling of the business process. The copyPayloadFromTask Assign activities copy the payload back to its source.

1. Double-click VacationRequest.bpel.
2. Expand the taskSwitch Switch activity.
3. Drag and drop an Assign activity below the copyPayloadFromTask Assign activity in the <case Task outcome is APPROVE> section of the Switch activity.
4. Double-click the Assign icon to display the Assign window.
5. Click the General tab.
6. Enter assignVacationApproval1 in the Name field.
7. Click Apply.
8. Click the Copy Operation tab.
9. Click Create and select Copy Operation.
10. Enter the following details:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>From Type</td>
<td>Expression</td>
</tr>
<tr>
<td>From Expression</td>
<td>string('Approved')</td>
</tr>
<tr>
<td>To Type</td>
<td>Variable</td>
</tr>
<tr>
<td>Variables</td>
<td>Expand and select Variables &gt; outputVariable &gt; payload &gt; client:VacationRequestProcessResponse &gt; client:result</td>
</tr>
<tr>
<td>Note:</td>
<td>The namespace number values (for example, client, ns1) can vary. Use the namespace values that automatically appear.</td>
</tr>
</tbody>
</table>

11. Click OK to close the Create Copy Operation window and the Assign window.
12. Repeat Steps 3 through 11 to create an Assign activity below the `copyPayloadFromTask` Assign activity in the `<case Task outcome is REJECT>` section. Enter the same details as described above, with the following exceptions:
   - Name it assignVacationApproval2
   - Set the Expression field to `string('Rejected')`

13. Repeat Steps 3 through 11 to create an Assign Activity below the `copyPayloadFromTask Assign` activity in the `<otherwise>` section. Enter the same details as described above, with the following exceptions:
   - Name it assignVacationApproval3
   - Set the Expression field to `string('Rejected')`

14. The process looks as follows:

15. Select Save from the File main menu.

16. Click the - sign to close the taskSwitch Switch activity.

Validating, Compiling, and Deploying the Order Approval Process
1. Go to the Application Navigator section.
2. Right-click VacationApproval.
3. Select Deploy > `my_integration_server_connection` > Deploy to default domain.
   This compiles the BPEL process. Check for errors by clicking the buttons at the bottom of the window. If there are no errors, deployment was successful.

Running the Order Approval Process
1. Log into Oracle BPEL Control by selecting Start > All Programs > Oracle - Oracle_Home > Oracle BPEL Process Manager > BPEL Control.
   The Dashboard tab of Oracle BPEL Control appears.
2. Enter the following details to log into Oracle BPEL Control and click Login.
3. Click VacationApproval in the Deployed BPEL Processes list.
4. Enter jcooper for the creator of the vacation.
5. Enter appropriate values for the remaining fields.
6. Click Post XML Message.
   The BPEL Processes tab displays a message similar to the following:
   Test Instance Initiated
7. Click the Instances tab at the top.
8. Click the OrderApproval instance.
   A message appears indicating that the instance is active.
9. Select Start > All Programs > Oracle - Oracle_Home > Oracle BPEL Process Manager > Sample Worklist Application to access the login window for Oracle BPEL Worklist Application:
10. Log in as jstein/welcome1.
    The user jstein is the manager of jcooper. This displays Oracle BPEL Worklist Application. A task waiting to be approved appears.
11. Select Claim in the Actions list for the task to approve.
12. Click Go.
    The task details and payload information appear.
13. Review the information. For example, the following information appears if you copied and pasted in the contents of OrderBookingPO_1.xml.
14. Select Approve from the Task Action list and click Go.
15. Log out as user jcooper.
16. Log into Oracle BPEL Worklist Application as jstein/welcome1.
17. Select Approve from the Actions list and click Go.
    After processing, no tasks appear in Oracle BPEL Worklist Application.
18. Log out.
19. Return to Oracle BPEL Control.
20. Click the Instances tab at the top.
21. Click the VacationApproval instance.
    A message appears indicating that the instance has completed.
22. Click the Audit and Flow links to observe additional details about the completed OrderApproval process.
Workflow Services

Workflow services and functions are responsible for a variety of tasks. This section describes the responsibilities of the following workflow services:

- EJB, SOAP, and Java Support for the Workflow Services
- Security Model for Services
- Task Service
- Task Query Service
- Identity Service
- Notification Service
- Task Metadata Service
- User Metadata Service
- Runtime Config Service

See Also: "Workflow Services Components" on page 15-6

EJB, SOAP, and Java Support for the Workflow Services

Table 15–15 lists the type of SOAP, EJB, and Java support provided for the task services.

<table>
<thead>
<tr>
<th>Service Name</th>
<th>Supports SOAP Web Services</th>
<th>Supports Remote EJB</th>
<th>Supports Local EJB</th>
<th>Supports Plain Java APIs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task Service</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Task Query Service</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Task Metadata Service</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Task Reports Service</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>User Metadata Service</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Runtime Config Service</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Identity Service:

- BPM Authentication Service | Yes | Yes
- BPM Authorization Service | Yes | Yes

Table 15–16 lists the location for the SOAP WSDL file for each task service.

<table>
<thead>
<tr>
<th>Service Name</th>
<th>SOAP WSDL Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task Service</td>
<td><a href="http://host:port/integration/services/TaskServicePort?WSDL">http://host:port/integration/services/TaskServicePort?WSDL</a></td>
</tr>
<tr>
<td>Task Metadata Service</td>
<td><a href="http://host:port/integration/services/TaskMetadataServicePort?WSDL">http://host:port/integration/services/TaskMetadataServicePort?WSDL</a></td>
</tr>
<tr>
<td>Task Query Service</td>
<td><a href="http://host:port/integration/services/TaskQueryServicePort?WSDL">http://host:port/integration/services/TaskQueryServicePort?WSDL</a></td>
</tr>
</tbody>
</table>
Workflow Services

<table>
<thead>
<tr>
<th>Service name</th>
<th>SOAP WSDL location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Runtime Config Service</td>
<td><a href="http://host:port/integration/services/RuntimeConfigService?WSDL">http://host:port/integration/services/RuntimeConfigService?WSDL</a></td>
</tr>
<tr>
<td>Identity Service</td>
<td><a href="http://host:port/integration/services/IdentityService?WSDL">http://host:port/integration/services/IdentityService?WSDL</a></td>
</tr>
<tr>
<td>Notification Service</td>
<td><a href="http://host:port/integration/services/NotificationService?WSDL">http://host:port/integration/services/NotificationService?WSDL</a></td>
</tr>
</tbody>
</table>

Security Model for Services

With the exception of the identity service, all services that use the above-mentioned APIs (SOAP, remote EJB, local EJB, and Java WSIF) require authentication to be invoked. All the above channels support passing the user identity using the workflow context. The workflow context contains either of the following:

- Login and password
- Token

The task query service exposes the authenticate operation that takes the login and password and returns the workflow context used for all services. Optionally, with each request, you can pass the workflow context with the login and password.

The authenticate operation also supports the concept of creating the context on behalf of a user with the admin ID and admin password. This enables you to create the context for a logged-in user to the Oracle BPEL Worklist Application if the password for that user is not available.

Security in SOAP Web Services

SOAP Web services also support Web service security. When Web service security is used, the workflow context does not need to be present in the SOAP input. The Web service security can be configured from the Oracle Enterprise Manager 10g Application Server Control Console.

Note: Workflow service SOAP clients cannot be used when Web service security is used.

See Also: "Configuring Single Sign-on Using SAML" in the Oracle Application Server Web Services Security Guide for details about propagating the identity of a user from a Web application to the Web service

Security in EJBs

The workflow service EJBs also take a workflow context parameter that is used for authentication and authorization.
Creating Workflow Context on Behalf of a User

The authenticate API operation on the task query service can create the workflow context on behalf of a user by passing the user ID and password of an admin user in the request. An admin user is a user who has the BPMWorkflowAdmin role. This created context is as if it was created using the password on behalf of the user.

In this example, the workflow context is created for user jcooper.

```java
ITaskQueryService taskQueryService = ...;
String realm = ...;
IWorkflowContext wCtx =
    taskQueryService.authenticate('bpeladmin', 'welcome1', realm, 'jcooper');
```

Task Service

The task service exposes operations to act on tasks. Table 15–17 describes the operations of the task service. Package oracle.bpel.services.workflow.task corresponds to the task service.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>acquireTask</td>
<td>Acquire a task.</td>
</tr>
<tr>
<td>acquireTasks</td>
<td>Acquire a set of tasks.</td>
</tr>
<tr>
<td>addAttachment</td>
<td>Add an attachment to a task.</td>
</tr>
<tr>
<td>addComment</td>
<td>Add a comment to a task.</td>
</tr>
<tr>
<td>delegateTask</td>
<td>Delegate a task to a different user. Both the current assignee and the user to whom the task is delegated can view and act on the task.</td>
</tr>
<tr>
<td>errorTask</td>
<td>Cause the task to error. This operation is typically used by the error assignee.</td>
</tr>
<tr>
<td>escalateTask</td>
<td>Escalate a task. The default escalation is to the manager of the current user. This can be overridden using escalation functions.</td>
</tr>
<tr>
<td>getApprovers</td>
<td>Get the previous approvers of a task.</td>
</tr>
<tr>
<td>getFutureParticipants</td>
<td>Get the future participants of a task. The future participants are returned in the form of a routing slip that contains simple participants — (participant node and parallel nodes that contain routing slips in them).</td>
</tr>
<tr>
<td>getUsersToRequestInfoForTask</td>
<td>Get the users from whom a request for information can be requested.</td>
</tr>
<tr>
<td>initiateTask</td>
<td>Initiate a task.</td>
</tr>
<tr>
<td>mergeAndUpdateTask</td>
<td>Merge and update a task. Use this operation when a partial task should be updated. A partial task is one in which not all the task attributes are present. In this partial task, only the following task attributes are interpreted:</td>
</tr>
<tr>
<td></td>
<td>■ Task payload</td>
</tr>
<tr>
<td></td>
<td>■ Comments</td>
</tr>
<tr>
<td></td>
<td>■ Task state</td>
</tr>
<tr>
<td></td>
<td>■ Task outcome</td>
</tr>
<tr>
<td>overrideRoutingSlip</td>
<td>Override the routing slip of a task instance with a new routing slip. The current task assignment is nullified and the new routing slip is interpreted as its task is initiated.</td>
</tr>
</tbody>
</table>
### Task Service Methods (Cont.)

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pushBackTask</td>
<td>Push back a task to the previous approver or original assignees. The original assignees do not need to be the approver as they may have reassigned the task, escalated the task, and so on. The property pushbackAssignee in wf_config.xml controls whether the task is pushed back to the original assignees or the approvers.</td>
</tr>
<tr>
<td>reassignTask</td>
<td>Reassign a task.</td>
</tr>
<tr>
<td>reinitiateTask</td>
<td>Reinitiate a task. Reinitiating a task causes a previously completed task to be carried forward so that the history, comments, and attachments are carried forward in a new task.</td>
</tr>
<tr>
<td>releaseTask</td>
<td>Release a previously acquired task.</td>
</tr>
<tr>
<td>releaseTasks</td>
<td>Release a set of previously acquired tasks.</td>
</tr>
<tr>
<td>removeAttachment</td>
<td>Remove a task attachment.</td>
</tr>
<tr>
<td>renewTask</td>
<td>Renew a task to extend the time it takes to expire.</td>
</tr>
<tr>
<td>requestInfoForTask</td>
<td>Request information for a task.</td>
</tr>
<tr>
<td>requestInfoForTaskWithReapproval</td>
<td>Request information for a task with reapproval. For example, assume jcooper created a task and jstein and wfaulk approved the task in the same order. When the next approver, cdickens, requests information with reapproval from jcooper, and jcooper submits the information, jstein and wfaulk approve the task before it comes to cdickens. If cdickens requests information with reapproval from jstein, and jstein submits the information, wfaulk approves the task before it comes to cdickens.</td>
</tr>
<tr>
<td>resumeTask</td>
<td>Resume a task. Operations can only be performed by the task owners (or users with the BPMWorkflowSuspend privilege) to remove the hold on a workflow. After a workflow is resumed, actions can be performed on the task.</td>
</tr>
<tr>
<td>resumeTasks</td>
<td>Resume a set of tasks.</td>
</tr>
<tr>
<td>routeTask</td>
<td>Allow a user to route the task in an adhoc fashion to the next user(s) who must review the task. The user can specify to route the tasks in sequential, parallel, or simple assignment. Routing a task is permitted only when the workflow permits adhoc routing of the task.</td>
</tr>
<tr>
<td>skipCurrentAssignment</td>
<td>Skip the current assignment and move to the next assignment or pick the outcome as set by the previous approver if there are no more assignees.</td>
</tr>
<tr>
<td>submitInfoForTask</td>
<td>Submit information for a task. This action is typically performed after the user has made the necessary updates to the task or has added comments or attachments containing additional information.</td>
</tr>
<tr>
<td>suspendTask</td>
<td>Allows task owners (or users with the BPMWorkflowSuspend privilege) to put a workflow on hold temporarily. In this case, task expiration and escalation do not apply until the workflow is resumed. No actions are permitted on a task that has been suspended (except resume and withdraw).</td>
</tr>
<tr>
<td>suspendTasks</td>
<td>Suspend a set of tasks.</td>
</tr>
<tr>
<td>updateOutcomeOfTasks</td>
<td>Update the outcome of a set of tasks.</td>
</tr>
<tr>
<td>updateTask</td>
<td>Update the task.</td>
</tr>
<tr>
<td>updateTaskOutcome</td>
<td>Update the task outcome.</td>
</tr>
</tbody>
</table>
The task query service queries tasks based on a variety of search criterion such as keyword, category, status, business process, attribute values, history information of a task, and so on. Table 15–18 describes the operations of the task query service, including how to use the service over SOAP. Package oracle.bpel.services.workflow.query corresponds to the task query service.

### Table 15–18 Task Query Service Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>authenticate</td>
<td>Authenticates a user with the identity authentication service and passes back a valid IWorkflowContext object. Authentication can optionally be made on behalf of another user.</td>
</tr>
<tr>
<td>createContext</td>
<td>Creates a valid IWorkflowContext object from a preauthenticated HTTP request.</td>
</tr>
<tr>
<td>getWorkflowContext</td>
<td>Gets a workflow context with the specified context token.</td>
</tr>
<tr>
<td>destroyWorkflowContext</td>
<td>Cleans up a workflow context that is no longer needed. This method is typically used when a user logs out.</td>
</tr>
<tr>
<td>getTaskDetailsById</td>
<td>Gets the details of a specific task from the task's taskId property.</td>
</tr>
<tr>
<td>getTaskDetailsByNumber</td>
<td>Gets the details of a specific task from the task's task number property.</td>
</tr>
<tr>
<td>getTaskHistory</td>
<td>Gets the last of the task versions for the specified task ID.</td>
</tr>
<tr>
<td>getTaskVersionDetails</td>
<td>Gets the specific task version details for the specified task ID and version number.</td>
</tr>
</tbody>
</table>

**See Also:** Oracle BPEL Process Manager Workflow Services API Reference located in the SOA_Oracle_Home/bpel/docs/workflow directory
This section describes the identity service component of Oracle BPEL Process Manager. The identity service is a thin Web service layer on top of the Oracle Application Server 10g security infrastructure, namely OracleAS JAAS Provider.

### Identity Service

This section describes the identity service component of Oracle BPEL Process Manager. The identity service is a thin Web service layer on top of the Oracle Application Server 10g security infrastructure, namely OracleAS JAAS Provider.

**Table 15–18 (Cont.) Task Query Service Methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queryTasks</td>
<td>Returns a list of tasks that match the specified filter conditions. Tasks are listed according to the ordering condition specified (if any). The entire list of tasks matching the criteria can be returned or clients can execute paging queries, in which only a specified number of tasks in the list are retrieved. The filter conditions are as follows:</td>
</tr>
<tr>
<td></td>
<td>- <strong>assignmentFilter</strong> — Filters tasks according to whom the task is assigned, or who created the task. Possible values for the assignment filter are as follows:</td>
</tr>
<tr>
<td></td>
<td>- <strong>ADMIN</strong> — No filtering; returns all tasks regardless of assignment or creator.</td>
</tr>
<tr>
<td></td>
<td>- <strong>ALL</strong> — No filtering; returns all tasks regardless of assignment or creator.</td>
</tr>
<tr>
<td></td>
<td>- <strong>CREATOR</strong> — Returns tasks where the context user is the creator.</td>
</tr>
<tr>
<td></td>
<td>- <strong>GROUP</strong> — Returns tasks that are assigned to one of the groups of which the context user is a member.</td>
</tr>
<tr>
<td></td>
<td>- <strong>MY</strong> — Returns tasks that are assigned to the context user.</td>
</tr>
<tr>
<td></td>
<td>- <strong>MY_AND_GROUP</strong> — Returns tasks that are assigned to either the context user, or one of the groups of which they are a member.</td>
</tr>
<tr>
<td></td>
<td>- <strong>OWNER</strong> — Returns tasks where the context user is the task owner.</td>
</tr>
<tr>
<td></td>
<td>- <strong>PREVIOUS</strong> — Returns tasks the context user previously updated.</td>
</tr>
<tr>
<td></td>
<td>- <strong>REPORTEES</strong> — Returns tasks that are assigned to reportees of the context user.</td>
</tr>
<tr>
<td></td>
<td>- <strong>keywords</strong> — An optional search string. This only returns tasks where the string is contained in the task title, task identification key, or one of the task text flex fields.</td>
</tr>
<tr>
<td></td>
<td>- <strong>predicate</strong> — An optional <code>oracle.bpel.services.workflow.repos.Predicate</code> object that allows clients to specify complex, SQL-like query predicates.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> To use the task query service over SOAP, call <code>Predicate.enableXMLSerialization(true)</code> to make the predicate object serializable.</td>
</tr>
<tr>
<td>queryViewTasks</td>
<td>Returns a list of tasks according to the criteria in the specified view. The entire list or paged list of tasks can be returned. Clients can specify additional filter and ordering criteria to those in the view.</td>
</tr>
</tbody>
</table>

**See Also:** Oracle BPEL Process Manager Workflow Services API Reference located in the SOA_Oracle_Home\bpel\docs\workflow directory
Workflow Services

Oracle BPEL Process Manager Workflow Services

(JAZN), or any custom user repository. It enables authentication and authorization of
users and the lookup of user properties, roles, group memberships, and privileges.
Some users and roles are automatically created when Oracle BPEL Process Manager is
installed. Seeded users include:

- guest
- default
- bpeladmin
- oc4jadmin

The identity service predefines the following roles, which can be granted to users to
perform workflow-related operations:

- **PUBLIC**—This role is an implicit JAZN role; it does not need to be granted
  explicitly to any of the users. If any user can authenticate with the worklist, then
  they can see tasks assigned to them or groups they belong to and act on these
tasks.

- **BPMWorkflowReassign**—This role enables a user to reassign tasks to any other
  user in the organization. A manager can always delegate tasks to any users under
  him in the organization hierarchy without any Reassign privileges. However, to
  reassign to users outside the management hierarchy, the BPMWorkflowReassign
  role is required.

- **BPMWorkflowSuspend**—This role enables users to suspend a process. If a
  process is suspended, then the expiration time does not apply. When the process is
  resumed, the expiration date is recomputed. Users cannot suspend the workflow if
  the process designer has designated Suspend as a restricted action, even if the
  user has the BPMWorkflowSuspend role.

- **BPMWorkflowViewHistory**—In general, a user can see only the task assignment
  sequence as part of their worklist. This role enables a user to drill down further
  into the BPEL business process audit trail from the task approval sequence.

- **BPMWorkflowAdmin**—This role enables a user to perform system actions on any
  workflow in the system. This role does not allow you to change the outcome of the
  task (such as approve or reject); it only allows you to perform actions such as
  delegate, escalate, and suspend. Only the task assignee or the task owner can
  change the outcome of the task.

- **BPMSystemAdmin**—Both BPMWorkflowAdmin and BPMSystemAdmin have the
  same level of workflow privileges.

- **BPMDefaultDomainAdmin**—This role provides a user with access to the default
  domain through Oracle BPEL Control.

**Note:** The BPMPublic role can be used and explicitly granted to
each user if a third-party provider does not support an implicit
PUBLIC role.

- **BPMWorkflowReassign**—This role enables a user to reassign tasks to any other
  user in the organization. A manager can always delegate tasks to any users under
  him in the organization hierarchy without any Reassign privileges. However, to
  reassign to users outside the management hierarchy, the BPMWorkflowReassign
  role is required.

- **BPMWorkflowSuspend**—This role enables users to suspend a process. If a
  process is suspended, then the expiration time does not apply. When the process is
  resumed, the expiration date is recomputed. Users cannot suspend the workflow if
  the process designer has designated Suspend as a restricted action, even if the
  user has the BPMWorkflowSuspend role.

- **BPMWorkflowViewHistory**—In general, a user can see only the task assignment
  sequence as part of their worklist. This role enables a user to drill down further
  into the BPEL business process audit trail from the task approval sequence.

- **BPMWorkflowAdmin**—This role enables a user to perform system actions on any
  workflow in the system. This role does not allow you to change the outcome of the
  task (such as approve or reject); it only allows you to perform actions such as
  delegate, escalate, and suspend. Only the task assignee or the task owner can
  change the outcome of the task.

- **BPMSystemAdmin**—Both BPMWorkflowAdmin and BPMSystemAdmin have the
  same level of workflow privileges.

- **BPMDefaultDomainAdmin**—This role provides a user with access to the default
domain through Oracle BPEL Control.

**See Also:** Oracle BPEL Process Manager Administrator’s Guide for
instructions on configuring the identity service and additional details
about the BPMSystemAdmin and BPMDefaultDomainAdmin roles.
Some of these roles are nested. The BPMWorkflowReassign, BPMWorkflowSuspend, and BPMWorkflowViewHistory roles are granted to the BPMWorkflowAdmin role. The BPMSystemAdmin role is granted to the seeded bpeladmin user.

The following table represents the relationship between the grantees and roles:

<table>
<thead>
<tr>
<th>Role/Grantee</th>
<th>bpeladmin</th>
<th>default</th>
<th>guest</th>
<th>BPMWorkflowAdmin</th>
<th>BPMSystemAdmin</th>
</tr>
</thead>
<tbody>
<tr>
<td>BPMSystemAdmin</td>
<td>Directly</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>BPMWorkflowAdmin</td>
<td>Indirectly through BPMSystemAdmin</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>Directly</td>
</tr>
<tr>
<td>BPMWorkflowReassign</td>
<td>Indirectly through BPMSystemAdmin</td>
<td>--</td>
<td>--</td>
<td>Directly</td>
<td>Indirectly through BPMWorkflowAdmin</td>
</tr>
<tr>
<td>BPMWorkflowSuspend</td>
<td>Indirectly through BPMSystemAdmin</td>
<td>--</td>
<td>--</td>
<td>Directly</td>
<td>Indirectly through BPMWorkflowAdmin</td>
</tr>
<tr>
<td>BPMWorkflowViewHistory</td>
<td>Indirectly through BPMSystemAdmin</td>
<td>--</td>
<td>--</td>
<td>Directly</td>
<td>Indirectly through BPMWorkflowAdmin</td>
</tr>
<tr>
<td>BPMDefaultDomainAdmin</td>
<td>Indirectly through BPMSystemAdmin</td>
<td>Directly</td>
<td></td>
<td>--</td>
<td>Directly</td>
</tr>
</tbody>
</table>

Creating Users and Groups
You use directory-specific tools to create realms, users, or groups. For example:

- To create users and groups when using OID, you use the Oracle Delegated Administration Services tools. See Oracle Identity Management Guide to Delegated Administration for more information.
- To create user and group credentials when using the XML-based JAZN provider, you use the JAZN Admintool to modify the jazn-data.xml file. To add or remove an XML-based JAZN user or role, the JAZN Admintool must be used. You can manually edit the users-properties.xml file to specify detailed user properties that JAZN does not support.

For example, to add a user to a specified realm, issue the following command:

```
java -jar jazn.jar -user adminUser -password adminPassword
-adduser realName newUser newUserPassword
```

The JAZN Admintool provides different command options. You can list all the options and their syntax with the -help option, as in:

```
java -jar jazn.jar -help
```

- If you are using a third-party LDAP server or a custom user repository, you must use the specific tools available for that directory.

See Also: Oracle Containers for J2EE Security Guide for instructions on using the JAZN Admintool

Identity Service Providers
Oracle BPEL Process Manager identity service supports three types of providers: JAZN, third-party LDAP, or custom plug-in, as shown in Figure 15-36.
The identity service providers perform the following operations:

- **Authentication**—authenticates users given their username and password
- **Authorization**—determines roles and group memberships for a specific user. These roles are then used to control access to various work items and operations on the worklist.
- **Retrieve user properties**—includes contact information such as first name, last name, phone, e-mail, preferred notification channel, language preference, time zone, and organization details such as manager name and reportees.

**The JAZN Provider** The JAZN provider mode, which is preconfigured, delegates all authentication and authorization inquiries to the JAZN layer. Two JAAS providers are supplied as part of the OC4J security infrastructure: the XML-based file and LDAP-based OID.

- **XML-Based JAZN Provider Type** — The XML-based provider type is used for lightweight storage of information in the XML files. All the user names, roles, and permissions are stored in XML files. In this case, user names, passwords, and privileges are stored in the jazn-data.xml file. In addition, Oracle BPEL Process Manager uses a user-properties.xml file that works in conjunction with this file to store detailed user properties such as name, e-mail, phone, and manager.

- **LDAP-Based JAZN Provider Type (Oracle Internet Directory)** — The LDAP-based provider type is based on the Lightweight Directory Access Protocol (LDAP) for centralized storage of information in a directory. OID is a standard LDAP-based directory that provides a single, centralized repository for all user data. It allows sites to manage user identities, roles, authorization, and authentication credentials, as well as application-specific preferences and profiles in a single repository.

**Third-Party LDAP Server** The third-party LDAP provider mode enables identity service to work with third-party LDAP servers such as Sun Directory Server (iPlanet), Microsoft Active Directory, or openLDAP. In this mode, identity service assumes that the directory is the central repository of all user data, including authentication credentials, roles, and profiles. The standard organizationalPerson, inetOrgPerson objects from the LDAP schema retrieve these details.

**See Also:**  
Oracle Application Server Containers for J2EE User’s Guide

**Custom User Repository Plug-ins** This mode enables you to plug in a non-LDAP-based user repository by registering a custom identity service provider. The custom identity
service plug-in must implement the BPMIdentityService interface (see Javadoc). This identityservice class name must be registered in is_config.xml.

See Also:

- "User and Role Properties" on page 15-106 for more information.
- Identity service configuration instructions in Oracle BPEL Process Manager Administrator’s Guide
- See SOA_Oracle_Home\bpe\docs\workflow\oracle\tip\pc\services\id entity for Javadoc on the BPMIdentityService interface

User and Role Properties

The identity service supports the following user properties:

- Display name
- Given name, middle name, and last name
- Description
- Title
- E-mail address
- Telephone number
- Home phone number
- Mobile phone number
- Fax number
- Pager number
- Manager ID
- Owners (applies to groups and roles, but not users)
- Time zone
- Language preference (Java locale)
- Notification preference (preferred notification channel)

The preceding properties are optional for Oracle BPEL Process Manager users. However, some features, such as task notification, are not available if the contact information is not present in the directory or in the users-properties file for the JAZN XML-based provider. Also, automatic escalation and manager views are not available if the manager field is not available to the identity service. If the user is not listed among the owners of the group, they cannot modify the rule defined for the group.

See Also: The service configuration chapter of Oracle BPEL Process Manager Administrator’s Guide for instructions on defining group ownership

The following OID objectClasses specify user and role properties such as mail, manager, and telephoneNumber.

- top
- person
Workflow Services

- cn
- sn
- description
- telephoneNumber

  organizationalPerson
  - title
  - telephoneNumber
  - facsimileTelephoneNumber

  inetOrgPerson
  - uid
  - displayName
  - givenName
  - manager
  - mail
  - homePhone
  - mobile
  - pager
  - preferredLanguage

  orclUserV2
  - middleName
  - orclTimeZone
  - orclWorkflowNotificationPref

  groupOfUniqueNames
  - description
  - owner
  - uniqueMember

  orclGroup
  - displayName
  - mail

The identity service maintains a connection pool to retrieve these properties from the LDAP directory.

If you are using the XML-based JAZN provider, the same entries are represented as XML elements in the `users-properties.xml` file in `SOA_Oracle_Home\bpel\system\services\config`

Multirealm Support

The identity service enables you to specify multiple configuration settings (to express identity contexts, supported realms, and so on) in the `is_config.xml` file. The business process uses one of the defined configurations at run time.
The configuration must specify the realm name to enable a business process to resolve the context at run time. For the JAZN provider, the realm name must match one of supported JAZN realm names. Otherwise, a run time exception is thrown. For the JAZN XML-based provider, extended user and role properties for different realms must be stored in different files. For the LDAP provider, the realm name can be any unique name, while the context is defined by the LDAP URL, user search base, and role search base nodes in the LDAP server tree. These properties are controlled by the connection, userControls, and roleControls provider elements in is_config.xml.

If the is_config.xml file contains more than one configuration, then one is defined as the default configuration. The default context is used by the BPEL process if no specific context information is found at run time. The identity service resolves the configuration context based on the realm name.

See Also: The service configuration chapter of the Oracle BPEL Process Manager Administrator’s Guide for configuration instructions

Authentication, Authorization, and Identity Service Providers

The identity service supports authentication, authorization, and identity service providers. The identity service provider is the default pseudoservice provider. It must be defined for each configuration in the is_config.xml file. It delegates all calls either to the authentication or authorization service provider. By default, all three service providers share the same context setting defined in the identity provider.

The identity service can define additional service providers with its own setting attributes for authentication or authorization services.

If the provider service attribute is set to Authentication, the setting and the provider context are used only for all authentication calls for the configuration. If the provider service attribute is set to Authorization, the setting and provider context are used only for authorization calls.

See Also: The multiple service providers section of the service configuration chapter of the Oracle BPEL Process Manager Administrator’s Guide for an example of a configuration with two providers:

- The JAZN XML-based identity service provider is used for all calls, except authentication
- The custom plug-in provider is used only for authentication calls

Notification Service

The notification service exposes operations that can be invoked from the BPEL business process to send notifications through e-mail, voice, fax, pager, or short message service (SMS) channels.

See Also:
- “Notifications from Workflow Services” on page 15-80
- Chapter 14, “Oracle BPEL Process Manager Notification Service”
- Oracle BPEL Process Manager Administrator’s Guide for instructions on configuring notification service delivery channels
Task Metadata Service

Task metadata service exposes operations to retrieve metadata information related to a task. Table 15–19 describes these methods. Package oracle.bpel.services.workflow.metadata corresponds to the task metadata service.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getOutcomes</td>
<td>Get the permitted outcomes of a task. The outcomes are returned with their display values.</td>
</tr>
<tr>
<td>getResourceBundleInfo</td>
<td>Get the resource bundle information of the task. The resource bundle information contains the location and the name of the bundle.</td>
</tr>
<tr>
<td>getRestrictedActions</td>
<td>Get the actions that are restricted for a particular task.</td>
</tr>
<tr>
<td>getTaskAttributes</td>
<td>Get the task message attributes.</td>
</tr>
<tr>
<td>getTaskAttributesForTaskDef</td>
<td>Get the message attributes for a particular task definition.</td>
</tr>
<tr>
<td>getTaskDefinition</td>
<td>Get the task definition associated with the task.</td>
</tr>
<tr>
<td>getTaskDefinitionById</td>
<td>Get the task definition by the task definition ID.</td>
</tr>
<tr>
<td>getTaskDefinitionOutcome</td>
<td>Get the outcomes given the task definition ID.</td>
</tr>
<tr>
<td>getTaskDisplay</td>
<td>Get the task display for a task.</td>
</tr>
<tr>
<td>getTaskDisplayRegion</td>
<td>Get the task display region for a task.</td>
</tr>
<tr>
<td>getVersionTrackedAttributes</td>
<td>Get the task attributes that when changed causes a task version creation.</td>
</tr>
<tr>
<td>listTaskMetadata</td>
<td>List the task definitions in the system.</td>
</tr>
</tbody>
</table>

See Also: Oracle BPEL Process Manager Workflow Services API Reference located in the SOA_Oracle_Home\bpe\docs\workflow directory

User Metadata Service

The user metadata service provides methods for managing metadata specific to individual users and groups. It is used for getting and setting user worklist preferences, managing user custom views, and managing workflow rules for users and groups.

For most methods in the user metadata service, the authenticated user can query and update their own user metadata. However, they cannot update metadata belonging to other users.

In the case of group metadata (for example, workflow rules for groups), only a user designated as an owner of a group (or a BPMWorkflowAdmin user) can query and update the metadata for that group. However, a user that has been granted the BPMWorkflowAdmin role can query and update metadata for any user or group.

Table 15–20 describes the methods in the user metadata service. Package oracle.bpel.services.workflow.user corresponds to the user metadata service.
Table 15–20  User Metadata Service Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>setVacationInfo</td>
<td>Sets a date range over which the user is unavailable for the assignment of tasks. (Dynamic assignment functions do not assign tasks to a user that is on vacation.)</td>
</tr>
<tr>
<td>getVacationInfo</td>
<td>Retrieves the date range (if any) during which a user is unavailable for the assignment of tasks.</td>
</tr>
<tr>
<td>getRuleList</td>
<td>Retrieves a list of rules for a particular user or group.</td>
</tr>
<tr>
<td>getRuleDetail</td>
<td>Gets the details for a particular workflow rule.</td>
</tr>
<tr>
<td>createRule</td>
<td>Creates a new rule.</td>
</tr>
<tr>
<td>updateRule</td>
<td>Updates an existing rule.</td>
</tr>
<tr>
<td>deleteRule</td>
<td>Deletes a rule.</td>
</tr>
<tr>
<td>increaseRulePriority</td>
<td>Increases the priority of a rule by one. Rules for a user or group are maintained in an ordered list of priority. Higher priority rules (those closer to the head of the list) are executed before rules with lower priority. This method does nothing if this rule already has the highest priority.</td>
</tr>
<tr>
<td>decreaseRulePriority</td>
<td>Decreases the priority of a rule by one. This method does nothing if this rule already has the lowest priority.</td>
</tr>
<tr>
<td>getRuleSetInfo</td>
<td>Returns information relating to the Oracle Business Rules rule set being used to store the rules for a particular user or group. This is useful if a client wants to make use of the rules SDK directly for manipulating rules, rather than using the user metadata service.</td>
</tr>
<tr>
<td>getUserTaskViewList</td>
<td>Gets a list of the user task views that the user owns.</td>
</tr>
<tr>
<td>getGrantedTaskViewList</td>
<td>Gets a list of user task views that have been granted to the user by other users. Users can use granted views for querying lists of tasks, but they cannot update the view definition.</td>
</tr>
<tr>
<td>getStandardTaskViewList</td>
<td>Gets a list of standard task views that ship with the workflow service, and are available to all users.</td>
</tr>
<tr>
<td>getUserInboxView</td>
<td>Gets a special view to store configuration information, allowing users to personalize their main inbox list of tasks.</td>
</tr>
<tr>
<td>getUserTaskViewDetails</td>
<td>Gets the details for a single view.</td>
</tr>
<tr>
<td>createUserTaskView</td>
<td>Creates a new user task view.</td>
</tr>
<tr>
<td>updateUserTaskView</td>
<td>Updates an existing user task view.</td>
</tr>
<tr>
<td>deleteUserTaskView</td>
<td>Deletes a user task view.</td>
</tr>
<tr>
<td>updateGrantedTaskView</td>
<td>Updates details of a view grant made to this user by another user. Updates are limited to hiding or unhiding the view grant (hiding a view means that the view is not listed in the main inbox page of the worklist application), and changing the name and description that the granted user sees for the view.</td>
</tr>
<tr>
<td>getUserPreferences</td>
<td>Gets a list of user preferences for the user. User preferences are simple name-value pairs of strings. User preferences are private to each user (but can still be queried and updated by BPMWorkflowAdmin).</td>
</tr>
<tr>
<td>setUserPreferences</td>
<td>Sets the user preference values for the user. Any preferences that were previously stored and are not in the new list of user preferences are deleted.</td>
</tr>
</tbody>
</table>
Workflow Services

Runtime Config Service

The runtime config service provides methods for managing metadata used in the task service run time environment. It principally supports management of task payload flex field mappings.

The task object used by the task service contains a number of flex field attributes, which can be populated with information from the task payload. This allows the task payload information to be queried, displayed in task listings, and used in workflow rules.

The runtime config service allows administrators to create mappings between simple task payload attributes and these flex field attributes.

Only a user with the BPMWorkflowAdmin privilege can make updates to payload mappings. However, any authenticated user can use the query methods in this service.

An administrator can create attribute labels for the various flex field attributes. These attribute labels provide a meaningful label for the attribute (for example, a label Location may be created for the flex field attribute TextAttribute1). A given flex field attribute may have multiple labels associated with it. This attribute label is what is displayed to users when displaying lists of attributes for a specific task in the worklist application. The attribute labels for a specific task type can be determined by calling the getTaskAttributesForTaskDefinition method on the task metadata service.

When defining attribute labels, the following fields are automatically populated by the service. You do not need to specify values for these attributes when creating or updating attribute labels:

- Id
- CreatedDate
- WorkflowType
- Active

---

Table 15–20 (Cont.) User Metadata Service Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getPublicPreferences</td>
<td>Gets a list of public preferences for the user. Public preferences are similar to user preferences, except any user can query them. However, only the user that owns the preferences, or the BPMWorkflowAdmin, can update them. Public preferences are useful for storing application wide preferences (preferences can be stored under a dummy user name, such as MyAppPrefs).</td>
</tr>
<tr>
<td>setPublicPreferences</td>
<td>Sets the public preferences for the user.</td>
</tr>
</tbody>
</table>

See Also:
- Chapter 16, “Worklist Application” for details about the rule configuration and user preference pages
- Oracle BPEL Process Manager Administrator’s Guide for details on how to designate a user as a group owner
- Oracle BPEL Process Manager Workflow Services API Reference located in the SOA_Oracle_Home\bpel\docs\workflow directory

Oracle BPEL Process Manager Workflow Services 15-111
Valid values for the task attribute field are as follows:
- TextAttribute1 through TextAttribute10
- FormAttribute1 through FormAttribute5
- UrlAttribute1 through UrlAttribute5
- DateAttribute1 through DateAttribute5
- NumberAttribute1 through NumberAttribute5

Mappings can then be created between task payload fields and the attribute labels. For example, the payload field `customerLocation` can be mapped to the attribute label `Location`. Different task types can share the same attribute label. This allows payload attributes from different task types that have the same semantic meaning to be mapped to the same attribute label.

**Note:** Only payload fields that are simple XML types can be mapped.

The runtime config service also provides methods for querying the dynamic assignment functions supported by the server.

**Table 15–21** describes the methods in the runtime config service. Package `oracle.bpel.services.workflow.runtimeconfig` corresponds to the runtime config service.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetWorkflowPayloadMappings</td>
<td>Gets a list of all the flex field mappings for a particular workflow task definition.</td>
</tr>
<tr>
<td>CreateAttributeLabel</td>
<td>Creates a new attribute label for a particular task flex field attribute.</td>
</tr>
<tr>
<td>updateAttributeLabel</td>
<td>Updates an existing attribute label.</td>
</tr>
<tr>
<td>DeleteAttributeLabel</td>
<td>Deletes an existing attribute label.</td>
</tr>
<tr>
<td>getAttributeLabelUsages</td>
<td>Gets a list of attribute labels (either all attribute labels, or labels for a specific type of attribute) for which mapping (if any) the labels are currently used.</td>
</tr>
<tr>
<td>createPayloadMapping</td>
<td>Creates a new mapping between an attribute label and a task payload field.</td>
</tr>
<tr>
<td>deletePayloadMapping</td>
<td>Deletes an existing payload mapping.</td>
</tr>
<tr>
<td>getUserDynamicAssignmentFunctions</td>
<td>Returns a list of the dynamic assignment functions that can select a user that are implemented on this server.</td>
</tr>
<tr>
<td>getGroupDynamicAssignmentFunctions</td>
<td>Returns a list of the dynamic assignment functions that can select a group that are implemented on this server.</td>
</tr>
</tbody>
</table>
Internationalization of Attribute Labels

Attribute labels provide a method of attaching a meaningful label to a task flex field attribute. It can be desirable to present attribute labels that are translated into the appropriate language for the locale of the user.

To achieve this, you can add entries to the WorkflowLabels.properties resource property file, and associated resource bundles in other languages. This file exists in the SOA_Oracle_Home\bpel\system\services\config\wfresource directory.

Entries for flex field attribute labels must be of the form:

```
FLEX_LABEL.[label name]=Label Display Name
```

For instance, the entry for a label named Location is:

```
FLEX_LABEL.Location=Location
```

Note that adding entries to these files for attribute labels is optional. If no entry is present in the file, the name of the attribute label as specified using the API is used instead.

Configuring the Assignment Service

This section describes how to configure the assignment service.

This section contains the following topics:

- Dynamic Assignment Functions
- Dynamically Assigning Task Participants with the Assignment Service
- Custom Escalation Function

Dynamic Assignment Functions

Dynamic assignment functions select a particular user or group from either a group, or from a list of users or groups.

The selection is made according to criteria specific to the particular dynamic assignment function. The three dynamic assignment functions shown in Table 15–22 are included with Oracle BPEL Process Manager:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROUND_ROBIN</td>
<td>Picks each user or group in turn.</td>
</tr>
<tr>
<td>least-busy</td>
<td>Picks the user or group with the least number of tasks currently assigned to it.</td>
</tr>
</tbody>
</table>

See Also:

- "Dynamic Assignment Functions" on page 15-113 for additional details
- Chapter 16, "Worklist Application" for details about flex field mapping
- Oracle BPEL Process Manager Workflow Services API Reference located in the SOA_Oracle_Home\bpel\docs\workflow directory

Internationalization of Attribute Labels

Attribute labels provide a method of attaching a meaningful label to a task flex field attribute. It can be desirable to present attribute labels that are translated into the appropriate language for the locale of the user.

To achieve this, you can add entries to the WorkflowLabels.properties resource property file, and associated resource bundles in other languages. This file exists in the SOA_Oracle_Home\bpel\system\services\config\wfresource directory.

Entries for flex field attribute labels must be of the form:

```
FLEX_LABEL.[label name]=Label Display Name
```

For instance, the entry for a label named Location is:

```
FLEX_LABEL.Location=Location
```

Note that adding entries to these files for attribute labels is optional. If no entry is present in the file, the name of the attribute label as specified using the API is used instead.

Configuring the Assignment Service

This section describes how to configure the assignment service.

This section contains the following topics:

- Dynamic Assignment Functions
- Dynamically Assigning Task Participants with the Assignment Service
- Custom Escalation Function

Dynamic Assignment Functions

Dynamic assignment functions select a particular user or group from either a group, or from a list of users or groups.

The selection is made according to criteria specific to the particular dynamic assignment function. The three dynamic assignment functions shown in Table 15–22 are included with Oracle BPEL Process Manager:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROUND_ROBIN</td>
<td>Picks each user or group in turn.</td>
</tr>
<tr>
<td>least-busy</td>
<td>Picks the user or group with the least number of tasks currently assigned to it.</td>
</tr>
</tbody>
</table>

Oracle BPEL Process Manager Workflow Services 15-113
These functions all check a user’s vacation status. A user that is currently unavailable is not automatically assigned tasks.

These dynamic assignment functions can be called using the custom XPath functions in a BPEL process or task definition.

- `wfDynamicUserAssign`
- `wfDynamicGroupAssign`

These XPath functions must be called with at least two, and optionally more parameters:

- The name of the dynamic assignment function being called.
- The name of the group on which to execute the function (or a list of users or groups).
- (Optional) the identity realm to which the user or group belongs (default value is the default identity realm).
- Additional optional parameters specific to the dynamic assignment function. In the case of the `most-productive` assignment function, this is the length of time (in days) to calculate a user’s productivity. The two other functions do not use additional parameters.

In addition, workflow rules created for a group can use dynamic assignment functions to select a member of that group for reassignment of a task.

In addition to the three functions, a dynamic assignment framework is provided that allows you to implement and configure your own dynamic assignment functions.

### Implementing a Dynamic Assignment Function

To implement your own dynamic assignment function, write a Java class that implements one or both of the following interfaces:

- `oracle.bpel.services.workflow.assignment.dynamic.IDynamicUserAssignmentFunction`
- `oracle.bpel.services.workflow.assignment.dynamic.IDynamicGroupAssignmentFunction`

If your dynamic assignment function selects users, implement the first interface. If it selects groups, implement the second interface. If it allows the selection of both users and groups, implement both interfaces.

The two interfaces above both extend the interface `oracle.bpel.services.workflow.assignment.dynamic.IDynamicAssignmentFunction`.

Your Java class should also implement the methods in that interface.

These interfaces as shown below:

```java
public interface IDynamicAssignmentFunction {
    /**
     * Sets the initialization parameters required by the function (if any)
     * This function is called automatically by the DynamicAssignmentRegistry
     * on registration of a new function. Initialization parameters can be
```
Configuring the Assignment Service

* specified in the xml definition of the function in the dynamic assign
* config file.
* @param initParams Map of String parameter values keyed by String parameter
* names.
* @throws DynamicAssignmentException if implementation of method finds invalid
* parameters
*/
public void setInitParams( Map initParams ) throws DynamicAssignmentException;

/**
* Gets the name of this Dynamic Assignment Function
* @return String the name of the Dynamic Assignment Function
*/
public String getFunctionName();

/**
* Gets a description of this Dynamic Assignment Function
* @return String description of function
*/
public String getDescription();

public interface IDynamicGroupAssignmentFunction extends IDynamicAssignmentFunction
{
/**
* This method contains the implementation of the Assignment Function
* Given a group name, it will return a subgroup in that group,
* according to the assignment pattern implemented
* @return String name of group
* @param groupName String name of group to select group from
* @param realm String name of Identity Service realm the group belongs
* to. If realm is null, the default Identity Service realm will be used.
* @param parameters String[] optional array of parameter values.
* Use of parameter values is implementation-specific.
*/
public String getGroupAssignment( String groupName, String realm, String[] parameters )
throws DynamicAssignmentException;

/**
* This method contains the implementation of the Assignment Function
* Given an arbitrary list of groups, it will return a group in that
* list, according to the assignment pattern implemented
* @return String name of group
* @param groupNames List of groups to select from
* @param realm String name of Identity Service realm the groups belong
* to. If realm is null, the default Identity Service realm will be used.
* @param parameters String[] optional array of parameter values.
* Use of parameter values is implementation-specific.
*/
public String getGroupAssignment( List groupNames, String realm, String[] parameters )
throws DynamicAssignmentException;
}

public interface IDynamicUserAssignmentFunction extends IDynamicAssignmentFunction
{
/**
* This method contains the implementation of the Assignment Function
* Given a group name, it will return a user in that group,
* @param context Map of String parameter values keyed by String parameter
* names.
*/
public String getUserAssignment( String groupName, Map context ) throws DynamicAssignmentException;

/**
* This method contains the implementation of the Assignment Function
* Given an arbitrary list of groups, it will return a user in that group,
* according to the assignment pattern implemented
* @return String name of user
* @param groupNames List of groups to select from
* @param realm String name of Identity Service realm the groups belong
* to. If realm is null, the default Identity Service realm will be used.
* @param parameters String[] optional array of parameter values.
* Use of parameter values is implementation-specific.
*/
public String getUserAssignment( List groupNames, String realm, String[] parameters )
throws DynamicAssignmentException;
}
Configuring the Assignment Service

* according to the assignment pattern implemented

@Retention
@Inherited
@Target(\{ElementType\})
public @interface DynamicAssignment {

* @override

public String getUserAssignment( String groupName, String realm, String[] parameters )
throws DynamicAssignmentException;

/**
* This method contains the implementation of the Assignment Function
* Given an arbitrary list of users, it will return a user in that
* list, according to the assignment pattern implemented
* @return String username of user
* @param usernames List of usernames to select user from
* @param realm String name of Identity Service realm the users belong
* to. If realm is null, the default Identity Service realm will be used.
* @param parameters String[] optional array of parameter values.
* Use of parameter values is implementation-specific.
* @throws DynamicAssignmentException if error encountered looking up
* group, or checking users.
*/
public String getUserAssignment( List usernames, String realm, String[] parameters )
throws DynamicAssignmentException;

public static List getAvailableUsersFromGroup( String group, String realm )
throws DynamicAssignmentException

/**
* Method uses the specified group name to lookup the sub-groups belonging to
* that group using the identity service.
* @return List of String names of groups
*/
public static List getAvailableUsersFromGroup( String group, String realm )
throws DynamicAssignmentException

The dynamic assignment framework also provides the utility class
oracle.bpel.services.workflow.assignment.dynamic.DynamicAssignmentUtils.
This class provides a number of methods that are useful when implementing dynamic
assignment functions.
These include:

/**
* Method returns a list of users belonging to the specified group
* that are available (i.e. not on vacation etc.)
* @return List of String usernames of available users
* @param group - name of group to lookup users for
* @param parameters List of String usernames to check
* @param realm String name of Identity Service realm the users belong
* to. If realm is null, the default Identity Service realm will be used.
* @param parameters String[] optional array of parameter values.
* Use of parameter values is implementation-specific.
* @throws DynamicAssignmentException if error encountered looking up
* group, or checking users.
*/
public static List getAvailableUsersFromList( List usernames, String realm )
throws DynamicAssignmentException

*/

15-116 Oracle BPEL Process Manager Developer’s Guide
**Configuring Dynamic Assignment Functions**

Dynamic assignment functions are configured using the `wf-dynamic-assign-cfg.xml` file in the SOA_Oracle_Home\bpel\system\services\config directory.

Each dynamic assignment function must have an entry in this file, in the form of a `<function>` tag.

The function tag must contain two attributes:

- `name` — the name of the function.
- `classpath` — the classpath of the class that implements the function.

In addition, the function tag can optionally contain any number of `<property>` tags. These tags pass initialization parameters to the dynamic assignment function. Each property tag must contain a name attribute. The value of the property is specified in the body of the tag.

The property values specified in these tags are passed as a map (indexed by the value of the name attributes) to the `setInitParameters` method of the dynamic assignment functions.
Two of the functions have initialization parameters. These are:

- **ROUND_ROBIN** — The parameter MAX_MAP_SIZE specifies the maximum number of sets of users or groups for which the function can maintain ROUND_ROBIN counts. The dynamic assignment function holds a list of users and groups in memory for each group (or list of users and groups) on which it is asked to execute the ROUND_ROBIN function.

- **most-productive** — The parameter DEFAULT_TIME_PERIOD specified the length of time (in days) over which to calculate the user’s productivity. This value can be overridden when calling the most-productive dynamic assignment function. Use an XPath function by specifying an alternative value as the third parameter in the XPath function call.

### Configuring Display Names for Dynamic Assignment Functions

The runtime config service provides methods for returning a list of available user and group dynamic assignment functions. These functions return both the name of the function, and a user-displayable label for the function. The functions support localization of the display name, so that it displays in the appropriate language for the context user. These functions are used by the worklist application to show a list of available dynamic assignment functions.

To specify display names (and appropriate translations) for your dynamic assignment functions, add entries to the resource property file WorkflowLabels.properties, and associated resource property files in other languages. This file exists in the SOA_Oracle_Home\bpel\system\services\config\wfresource directory.

Entries for dynamic assignment functions must be of the form:

```
DYN_ASSIGN_FN.[function name]=Function Display Name
```

For instance, the entry for the ROUND_ROBIN function is:

```
DYN_ASSIGN_FN.ROUND_ROBIN = Round Robin
```

Note that adding entries to these files for dynamic assignment functions is optional. If no entry is present in the file, then the name of the function (for example, ROUND_ROBIN) is used instead.

### Dynamically Assigning Task Participants with the Assignment Service

Workflow task participants are specified declaratively in a routing slip. The routing slip guides the workflow by specifying the participants and how they participate in the workflow task (for example, management chain hierarchy, sequential list of approvers, and so on).

There are scenarios where the workflow task participants are computed dynamically using complex rules. To support such dynamic assignment, an assignment service is used. The assignment service is responsible for determining the task assignees. You can also implement your own assignment service and plug in that implementation for use with a particular workflow.

This section contains the following topics:

- Assignment Service Overview
- Implementing an Assignment Service
- Example of Assignment Service Implementation
- Deploying a Custom Assignment Service
Assignment Service Overview
The assignment service determines the following task assignment details in a workflow:

- The assignment when the task is initiated
- The assignment when the task is reinitiated
- The assignment when a user updates the task outcome. When the task outcome is updated, the task can either be routed to other users or completed.
- The assignees from whom information for the task can be requested
- If the task supports reapproval from the Oracle BPEL Worklist Application, a user can request information for reapproval.
- The users who reapprove the task if reapproval is supported.

The workflow service identifies and invokes the assignment service for a particular task to determine the task assignment.

For example, a simple assignment service iteration is as follows:

1. A client initiates an expense approval task whose routing is determined by the assignment service.
2. The assignment service determines that the task assignee is jcooper.
3. When jcooper approves the task, the assignment service assigns the task to jstein. The assignment service also specifies that a notification must be sent to the creator of the task, jlondon.
4. jstein approves the task and the assignment service indicates that there are no more users to which to assign the task.

Implementing an Assignment Service
The assignment service is implemented with the IAssignmentService interface.

The workflow service passes the following information to the assignment service to determine the task assignment:

- Task document — The task document that is executed by the workflow. The task document contains the payload and other task information like current state, and so on.
- Map of properties — When an assignment service is specified, a list of properties can also be specified to correlate callbacks with backend services that determine the task assignees.
- Task history — The task history is a list of chronologically ordered task documents to trace the history of the task. The task documents in this list contain a subset of attributes in the actual task (such as state, updatedBy, outcome, updatedDate, and so on).
Example of Assignment Service Implementation

Notes:

- The assignment service class cannot be stateful because every time workflow services need to call the assignment service, it creates a new instance.
- The getAssigneesToRequestForInformation method can be called multiple times because one of the criteria to show the request-for-information action is that there are users to request information. Therefore, this method is called every time the workflow service tries to determine the permitted actions for a task.

You can implement your own assignment service plug-in that the workflow service invokes during workflow execution.

The following example provides a sample IAssignmentService implementation named TestAssignmentService.java.

```java
package oracle.bpel.services.workflow.test.workflow;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import oracle.bpel.services.workflow.metadata.routingslip.model.*;
import oracle.bpel.services.workflow.metadata.routingslip.model.ParticipantsType;
import oracle.bpel.services.workflow.task.IAssignmentService;
import oracle.bpel.services.workflow.task.ITaskAssignee;
import oracle.bpel.services.workflow.task.model.Task;
public class TestAssignmentService implements oracle.bpel.services.workflow.task.IAssignmentService {
    static int numberOfApprovals = 0;
    static String[] users = new String[] {"jstein", "wfaulk", "cdickens"};
    public Participants onInitiation(Task task, Map propertyBag) {
        return createParticipant();
    }
}
```

Notes:

- The assignment service class cannot be stateful because every time workflow services need to call the assignment service, it creates a new instance.
- The getAssigneesToRequestForInformation method can be called multiple times because one of the criteria to show the request-for-information action is that there are users to request information. Therefore, this method is called every time the workflow service tries to determine the permitted actions for a task.
public Participants onReinitiation(Task task, 
Map propertyBag) {
    return null;
}

public Participants onOutcomeUpdated(Task task, 
Map propertyBag, 
String updatedBy, 
String outcome) {
    return createParticipant();
}

public Participants onAssignmentSkipped(Task task, 
Map propertyBag) {
    return null;
}

public List getAssigneesForRequestForInformation(Task task, 
Map propertyBag) {
    List rfiUsers = new ArrayList();
    rfiUsers.add("jcooper");
    rfiUsers.add("jstein");
    rfiUsers.add("wfaulk");
    rfiUsers.add("cdickens");
    return rfiUsers;
}

public List getReapprovalAssignees(Task task, 
Map propertyBag, 
ITaskAssignee infoRequestedAssignee) {
    List reapprovalUsers = new ArrayList();
    reapprovalUsers.add("jstein");
    reapprovalUsers.add("wfaulk");
    reapprovalUsers.add("cdickens");
    return reapprovalUsers;
}

private Participants createParticipant() {
    if (numberOfApprovals > 2) {
        numberOfApprovals = 0;
        return null;
    }

    String user = users[numberOfApprovals++];

    ObjectFactory objFactory = new ObjectFactory();
    Participants participants = objFactory.createParticipants();
    Participant participant = objFactory.createParticipantsTypeParticipant();
    participant.setName("Loan Agent");
    ResourceType resource2 = objFactory.createResourceType(user);
    resource2.setIsGroup(false);
    resource2.setType("STATIC");
    participant.getResource().add(resource2);
    participants.getParticipantOrSequentialParticipantOrAdhoc().add(participant);
    return participants;
}

Deploying a Custom Assignment Service

You must use one of the following methods to make an assignment service
implementation class and its related classes available in the class path of Oracle BPEL
Process Manager:
Load your classes in the SOA_Oracle_Home\bpel\system\classes directory and unzip your JAR files in the same directory.

Change the Oracle BPEL Process Manager shared library to include your JAR files.

---

**Note:**

- You cannot create different versions of the assignment service for use in different BPEL processes unless you change package names or class names.
- Java classes and JAR files in the suitcase are not available in the class path and therefore cannot be used as a deployment model for the assignment service.
- The steps must be repeated for each node in a cluster.

---

**Custom Escalation Function**

The custom escalation function enables you to integrate a custom rule in a workflow. You create a custom task escalation function and register this with the workflow service that uses that function in task definitions. The **Advanced Settings** section of the Human Task editor enables you to integrate the rule in a human task.

**See Also:** "Specifying Escalation Rules" on page 15-47 for details

---

**Workflow Service and Identity Service Related XPath Extension Functions**

Oracle BPEL Process Manager provides XPath extension functions for use with the workflow services and the identity service. XPath extension functions mimic XPath 2.0 standards. Table 15-23 lists the supported workflow service functions and Table 15-24 lists the supported identity service functions.

**Table 15–23  Workflow Service Functions**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>See</th>
</tr>
</thead>
<tbody>
<tr>
<td>hwf:clearTaskAssignees()</td>
<td>Clears the task assignees in a task.</td>
<td>&quot;clearTaskAssignees&quot; on page D-28</td>
</tr>
<tr>
<td>hwf:createWordMLDocument()</td>
<td>Creates a Word document by transforming the given XSLT to WordML.</td>
<td>&quot;createWordMLDocument&quot; on page D-29</td>
</tr>
<tr>
<td>hwf:getNumberOfTaskApprovals()</td>
<td>Gets the number of task approvals.</td>
<td>&quot;getNumberOfTaskApprovals&quot; on page D-30</td>
</tr>
<tr>
<td>hwf:getPreviousTaskApprover()</td>
<td>Gets the previous task approver.</td>
<td>&quot;getPreviousTaskApprover&quot; on page D-30</td>
</tr>
<tr>
<td>hwf:getTaskAttachmentByIndex()</td>
<td>Gets the task attachment by attachment index.</td>
<td>&quot;getTaskAttachmentByIndex&quot; on page D-30</td>
</tr>
<tr>
<td>hwf:getTaskAttachmentByName()</td>
<td>Gets the task attachment by attachment name.</td>
<td>&quot;getTaskAttachmentByName&quot; on page D-30</td>
</tr>
</tbody>
</table>

---
Workflow Service and Identity Service Related XPath Extension Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>See</th>
</tr>
</thead>
<tbody>
<tr>
<td>hwf:getTaskAttachmentContents()</td>
<td>Gets the task attachment contents by attachment name.</td>
<td>&quot;getTaskAttachmentContents&quot; on page D-31</td>
</tr>
<tr>
<td>hwf:getTaskAttachmentsCount()</td>
<td>Gets the number of task attachments.</td>
<td>&quot;getTaskAttachmentsCount&quot; on page D-31</td>
</tr>
<tr>
<td>hwf:getTaskAttachmentByIndex()</td>
<td>Gets the resource string for a particular task</td>
<td>&quot;getTaskAttachmentByIndex&quot; on page D-30</td>
</tr>
<tr>
<td>hwf:wfDynamicGroupAssign()</td>
<td>Gets the name of an identity service group, selected according to the specified assignment pattern.</td>
<td>&quot;wfDynamicGroupAssign&quot; on page D-32</td>
</tr>
<tr>
<td>hwf:wfDynamicUserAssign()</td>
<td>Gets the name of an identity service user, selected according to the specified assignment pattern.</td>
<td>&quot;wfDynamicUserAssign&quot; on page D-33</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>See</th>
</tr>
</thead>
<tbody>
<tr>
<td>ids:getDefaultRealmName()</td>
<td>Gets the default realm name.</td>
<td>&quot;getDefaultRealmName&quot; on page D-23</td>
</tr>
<tr>
<td>ids:getGroupProperty()</td>
<td>Gets a group property.</td>
<td>&quot;getGroupProperty&quot; on page D-24</td>
</tr>
<tr>
<td>ids:getManager()</td>
<td>Gets the manager of a given user.</td>
<td>&quot;getManager&quot; on page D-24</td>
</tr>
<tr>
<td>ids:getReportees()</td>
<td>Gets the direct reportees of the user.</td>
<td>&quot;getReportees&quot; on page D-25</td>
</tr>
<tr>
<td>ids:getSupportedRealms()</td>
<td>Gets the supported realm names.</td>
<td>&quot;getSupportedRealmNames&quot; on page D-25</td>
</tr>
<tr>
<td>ids:getUserProperty()</td>
<td>Gets a user property.</td>
<td>&quot;getUserProperty&quot; on page D-25</td>
</tr>
<tr>
<td>ids:getUserRoles()</td>
<td>Gets the user roles.</td>
<td>&quot;getUserRoles&quot; on page D-25</td>
</tr>
<tr>
<td>ids:getUsersInGroup()</td>
<td>Gets the users in a group.</td>
<td>&quot;getUsersInGroup&quot; on page D-26</td>
</tr>
<tr>
<td>ids:isUserInRole()</td>
<td>Verifies if a user has a given role.</td>
<td>&quot;isUserInRole&quot; on page D-27</td>
</tr>
<tr>
<td>ids:lookupGroup()</td>
<td>Gets the group object.</td>
<td>&quot;lookupGroup&quot; on page D-28</td>
</tr>
<tr>
<td>ids:lookupUser()</td>
<td>Gets the user object.</td>
<td>&quot;lookupUser&quot; on page D-28</td>
</tr>
</tbody>
</table>

### Deprecated Workflow Service and Identity Service Functions

Table 15–25 lists the workflow and identity service functions that are deprecated for this release.

<table>
<thead>
<tr>
<th>Workflow Function</th>
<th>Identity Service Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>ora:getNumberOfTaskApprovals()</td>
<td>ora:getGroupProperty()</td>
</tr>
</tbody>
</table>

Oracle BPEL Process Manager Workflow Services  15-123
NLS Configuration

You can specify resource bundles for displaying task details in different languages in Oracle BPEL Worklist Application.

In addition, the resource property file `WorkflowLabels.properties` can be used for setting display names for the following:

- Dynamic assignment functions
- Payload mapping attribute labels
- Task attributes

See Also:

- "Specifying Multilingual Settings" on page 15-48 for details about resource bundles
- "Internationalization of Attribute Labels" on page 15-113 and "Configuring Display Names for Dynamic Assignment Functions" on page 15-118 for additional details about the resource property file `WorkflowLabels.properties`.

Summary

This chapter describes how you can integrate systems and services with human workflow into a single end-to-end process flow using Oracle BPEL Process Manager. The predefined workflow participant types are described, as are the components of workflow services—the task service, task routing service, identity service, worklist service, notification service, and others.
The Oracle BPEL Worklist Application (Worklist Application) is a Web interface that enables users to act on their assigned human workflow tasks. This chapter discusses the sample Worklist Application that is provided with Oracle BPEL Process Manager, and how you can modify it to create your own worklist application.

This chapter contains the following topics:

- Use Cases for the Worklist Application
- Overview of Worklist Application Concepts
- Features of the Worklist Application
- Accessing the Worklist Application in Local Languages
- Customizing the Worklist Application
- Building Clients for Workflow Services
- Summary

Use Cases for the Worklist Application

You can use the Web interface of the Worklist Application for any activity that requires you to act on tasks in a BPEL process. A manager can approve employee vacation requests or a loan agent can review a loan application, each of which has been submitted as part of a BPEL process. Supervisors or group administrators can use the Worklist Application to analyze tasks assigned to the group and route them appropriately. Worklist Application users can also update payloads, attach documents or comments, and route the task to other users, in addition to completing tasks by providing conclusions such as approvals or rejections.

The Worklist Application is demonstrated in the following use cases:

- Vacation Request—In this use case, an employee files a vacation request that is routed to his manager for approval. The manager sees the task in the Worklist Application in the My Tasks tab.
- Document Review—In this use case, an author submits a document for review by multiple reviewers in parallel.
- Expense Request Approval—In this use case, an employee’s expense request is automatically routed to his manager for approval. The manager sees the task in the Worklist Application in the My Tasks tab. After the manager’s approval, the task may be routed further up the management chain, depending on the context of the expense request. The BPEL process uses a decide activity to define the approval chain for a human workflow task dynamically, using Oracle Business
Rules. The business rules determine the approval level required for the expense request, based on factors such as the amount of the request and the type of expense. The sample demonstrates how the approval chain for each expense report can be different, and how users can change the rules governing the approvals at run time.

■ Help Desk Service Request—In this use case, the supervisor resolves a service request by assigning it to any of his reportees using ad hoc routing. The assignee then approves the task after he responds to the service request.

■ Loan Demo Plus—In this use case, a loan application is assigned to the LoanAgent role. All loan agents see the task in their My & Group tasks view. One of the loan agents claims the task and reviews it. If the loan agent approves it, and if the loan amount is greater than $100,000, then the task is routed further, to two levels of management approval. When the loan agent’s managers log in to their worklists, they see tasks that were routed to them and the actions performed by the previous approvers (for example, suggested APR, comments, or attachments).

Sample applications that are built with the workflow service APIs and demonstrate common features such as listing, updates, approvals, and login and logout are also provided.

See: SOA_Oracle_Home\bpel\samples\demos for the following directories:

■ VacationRequest
■ DocumentReview
■ ExpenseReportApproval
■ HelpDeskServiceRequest
■ LoanDemoPlus

See: SOA_Oracle_Home\bpel\samples\ for the following:

■ demos\ExpenseReportApproval
■ demos\LoanDemoPlus
■ tutorials\132.UserTasks
■ utils\AsyncLoanService\StarLoanUI

The OrderBooking tutorial also demonstrates how to use the Worklist Application to approve a purchase order manually.

See: Oracle BPEL Process Manager Order Booking Tutorial

Overview of Worklist Application Concepts

Chapter 15, "Oracle BPEL Process Manager Workflow Services" discussed how BPEL workflow services enable you to interweave human interactions along with connectivity to systems and services into an end-to-end process flow. The workflow service provides a programmatic interface to view and manage tasks from the BPEL process. The tasks displayed depend on the user’s profile, and the actions allowed depend on the user’s privileges. This Worklist Application is layered on top of the BPEL workflow service.
Overview of Worklist Application Concepts

Worklist Application User Types

The Worklist Application recognizes different types of users, as listed in Table 16–1.

<table>
<thead>
<tr>
<th>Type of User</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>End user</td>
<td>Acts on tasks assigned to him or his group and has access to system and custom actions, routing rules, and custom views</td>
</tr>
<tr>
<td>Supervisor (manager)</td>
<td>Acts on the tasks, reports, and custom views of his reportees, in addition to his own end-user access</td>
</tr>
<tr>
<td>Process owner</td>
<td>Acts on tasks belonging to the process but assigned to other users, in addition to his own end-user access</td>
</tr>
<tr>
<td>Group administrator</td>
<td>Manages group rules and dynamic assignments, in addition to his own end-user access</td>
</tr>
<tr>
<td>Workflow administrator</td>
<td>Administers tasks that are in an errored state, for example, tasks that must be reassigned or suspended. The workflow administrator can also change application preferences and map flex fields, and manage rules for any user or group, in addition to his own end-user access.</td>
</tr>
</tbody>
</table>

See “Identity Service” on page 15-102 for more information about predefined roles in the identity service.

Task Components

A work item or task that is assigned to a user has the following components:

- **Task attributes**—Includes task title, number, status, priority, expiration, identification key, assignees, and other flex fields.
- **Task form**—Consists of detailed information (the payload) about the task; for example, a loan application in the LoanDemoPlus sample or support ticket details in the HelpDeskServiceRequest sample.
- **Task comments**—Comments entered by various users who have participated in the workflow.
- **Task attachments**—Other documents or reference URLs that are associated with a task. These are typically associated with the workflow by the BPEL process or attached and modified by any of the participants in the workflow.
- **Task history**—Consists of the approval sequence and the update history for the task. The history maintains an audit trail of the actions performed by the participants in the workflow and a snapshot of the task payload and attachments at various points in the workflow.

The types of actions that users can perform on a task include:

- **Update task details**—The task form can include content that needs to be added or modified by the task reviewer. The reviewer can modify the task priority, include comments, or add attachments to the task.
- **Change outcome for the task**—As part of the process model, the workflow designer can include various custom outcomes for the task (for example, approve or reject, acknowledge, defer). If a user modifies a task outcome, it is removed from his worklist and routed to the next approver or back to the business process based on the workflow pattern.
Features of the Worklist Application

- **Perform system actions**—In addition to the custom actions specified as part of workflow modeling, the user can perform other system actions such as escalate or delegate. These actions are available on all tasks based on the user’s privileges. The process owner or workflow administrator can always perform any of these operations on processes that they own. See “Task Actions” on page 16-10 for more information about system actions.

Features of the Worklist Application

Use Internet Explorer 6.0 or Mozilla Firefox 1.0.4 to access the Worklist Application.

1. Open a Web browser.
2. Go to the following URL:
   http://hostname:portnumber/integration/worklistapp/Login
   ■ hostname is the name of the host on which Oracle BPEL Process Manager is installed
   ■ The portnumber used at installation (typically 9700 or 8888) is noted in bpelsetupinfo.txt at SOA_Oracle_Home/install/

   You can also select Start, then All Programs, then Oracle - Oracle_Home, then Oracle BPEL Process Manager, and then Worklist Application.
3. Type the username and password, and click Login.
   You can use jstein and welcome1 to access the sample Worklist Application.
   The username and password must exist in the user community provided to JAZN. See Oracle BPEL Process Manager Administrator’s Guide for the organizational hierarchy of the demonstration user community used in examples throughout this chapter. See “Identity Service” on page 15-102 for information on JAZN.

   The Worklist Application displays tasks specific to the logged-in user based on the user’s permissions and assigned groups and roles. Figure 16–1 shows the Worklist Application for the user jstein, who is a manager and is responsible for approving or rejecting his reportees’ vacation requests.
Features of the Worklist Application

Figure 16–1 Worklist Application—Task Listing (Home) Page

All task interactions—listing tasks, viewing task details, reassigning tasks, performing actions on tasks, setting outcomes, and so on—are initiated from the Task Listing (home) page. As Figure 16–1 shows, when jstein logs in to the Worklist Application, he sees the Task Listing (home) page, which shows the tasks assigned to him and to the group to which he belongs. Because jstein is a manager, the My Staff Tasks tab also appears. For tasks assigned to jstein, he selects an action from the Actions list to participate in the workflow. For tasks assigned to a group to which jstein belongs, he must claim the task before selecting an action. The task is not available to other users until jstein releases it back to the group.

From the home page, you can retrieve worklist tasks by using the Search field to do a keyword search or by using the Category, Priority, and Status fields to specify search criteria. The category filters that are available depend on which tab is selected. From the My Tasks tab, the category filters are My, Group, My & Group, and Previous (tasks worked on in the past). From the My Staff Tasks tab, the only category filter is Reportees. From the Initiated Tasks tab, the only category filter is Creator. In addition to the My Tasks, My Staff Tasks, and Initiated Tasks tabs, other tabs may be displayed, depending on the role granted to the logged-in user, as described in Table 16–2 (Tabs). From the Administration Tasks tab, the category filter is Owner if the user (who has been granted the BPMWorkflowAdmin role in order to see this tab) owns the tasks and Admin otherwise.

Table 16–2 describes the salient features of the Task Listing (home) page of the Worklist Application shown in Figure 16–1.
### Features of the Worklist Application

#### Table 16–2  Contents of the Worklist Application *My Tasks* Page

<table>
<thead>
<tr>
<th>Location in Figure 16–1</th>
<th>Page Element</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Top left</strong></td>
<td><strong>Tabs</strong>—The tabs displayed depend on the role granted to the logged-in user. Everyone sees <em>My Tasks</em> and <em>Initiated Tasks</em>. Managers also see <em>My Staff Tasks</em>. A user with the BPMWorkflowAdmin role also sees the <em>Administration Tasks, Manage Rules, Flex Field Mappings, and Application Customization</em> tabs. See “Using the Administration Functions” on page 16-27 for more information.</td>
</tr>
<tr>
<td></td>
<td><em>Welcome jstein [jazn.com]—In the banner area, the logged-in user’s name appears. Click the user name to display information. See “User and Group Information” on page 16-39 for more information.</em></td>
</tr>
<tr>
<td><strong>Top right</strong></td>
<td><strong>Reports link</strong>—The following reports are available: Unattended Tasks Report, Tasks Priority Report, Tasks Cycle Time Report, Tasks Productivity Report. See “Creating Reports” on page 16-33 for more information.</td>
</tr>
<tr>
<td></td>
<td><strong>Preferences link</strong>—Manages the logged-in user’s preferences, including setting vacation and other workflow rules, managing custom views, and customizing worklist displays. See “Setting Preferences” on page 16-21 for more information.</td>
</tr>
<tr>
<td><strong>Left pane</strong></td>
<td><strong>Work Queues</strong>—Standard, custom, and proxy views. See “Using Work Queues” on page 16-20 for more information.</td>
</tr>
</tbody>
</table>
This section contains the following topics:

- Using the Task Details Page
- Using Advanced Search
- Viewing a Bar Chart of Task Status
- Using Work Queues
- Setting Preferences
- Using the Administration Functions
Features of the Worklist Application

- Creating Reports
- User and Group Information

Using the Task Details Page

If you click a task in the Task Title column, the Task Details page for that task is displayed, as shown in Figure 16–2.
Features of the Worklist Application

Figure 16–2 Task Details Page

- **Task Number:** 19023
- **Title:** Help desk request for mtvan
- **State:** Assigned
- **Outcome:** Updated
- **Created Date:** 08/22/06
- **Expiration Date:** 08/23/06 01:32 PM
- **Creator:** mtvan
- **Acquired By:** mtvan
- **Assignee:** jten(0)

- **Requester:** mtvan
- **First Name:** Tuan
- **Last Name:** Tuan
- **Email:** jten1@burl24.asi.doing
- **Phone:** jten

- **Location:** CA
- **Type:** Hardware
- **Problem Description:** Laptop battery health
- **Severity:** 2
- **Status:** Open

- **Resolution:**
- **Comment:**
- **Resolved By:**

- **Short History:** Task Created, Assigned, Updmtvan, Updated By mtvan, Updated Date Aug 22, 2006 01:32 PM
This section describes the following elements of the Task Details page:

- Task Actions
- Request Status
- Header Section
- Payload Section
- Comments and Attachments Section
- History Section
- Routing
- Requesting More Information
- Reassignment
- Parallel Tasks
- Determining Action Permissions

**Task Actions**

Figure 16–3 shows a Task Action list. The actions in the list depend on the task design, the state of the task (for example, if the task has been completed, then no actions are listed), and the roles assigned to the logged-in user. Custom actions (actions defined in the BPEL process), such as Accept or Reject, are listed first. System actions, such as Escalate or Suspend, are listed below a separator line.

![Task Actions List](image)

You act on tasks using either the Task Action list or a button. The Task Action list contains actions that do not require additional input, such as accept, reject, renew, and suspend. Buttons are provided for actions that require additional input, such as reassignment and requests for information.

System actions are available on all tasks based on the user’s privileges. Table 16–3 lists system actions.
After you select one of the actions, the task is routed to the next step, depending on how the business process was designed. When a task is completed, all actions and form elements are disabled.

**Request Status**

For every update request (custom or system action) that you submit, the status of the request is displayed in the left portion of the header. If a request is successful, then you see a confirming message, as shown near the top of the page in Figure 16–4.
If a request is not successful, then you see an error message, as shown in Figure 16–5. You can click the link for additional information about the error.

The error shown in Figure 16–5 occurs when a user has attempted an action that is not permitted. This is possible in the following scenarios:

- The task expired between the time the user loaded the page and actually performed the action.
- The task was claimed and updated concurrently by another user (such as a manager, owner, or administrator) between the time the user loaded the page and actually performed the action.
Errored tasks are not assigned to a specific user. They are only assigned to the bpeladmin user. If you are a user other than bpeladmin and want to see these errors, select All in the Category list and Errored or Any in the Status list.

**Header Section**

Figure 16–6 shows the header section. Header information includes the task number and title; the state, outcome, and priority of the BPEL process, and information about who created, updated, claimed, or is assigned to the task. It also displays dates related to task creation, last modification, and expiration.

![Figure 16–6 Header Section of the Task Details Page](image)

**Payload Section**

Figure 16–7 shows the payload section for the Vacation Request Process Request workflow. The fields displayed—Creator, From Date, To Date, Reason—reflect how the BPEL process for vacation approval was designed, using the autogenerated JSP.

![Figure 16–7 Payload Section](image)

See "Automatically Generating a Simple Task Display Form" on page 15-68 for information on using the autogenerated JSP in your workflow design.

**Comments and Attachments Section**

Figure 16–8 shows where you add or delete comments and attachments. To add or delete a comment or attachment, you must have permission to update the task.
Features of the Worklist Application

**Figure 16–8  Adding a Comment or Attachment**

A newly added comment and the comment writer’s username are appended to the existing comments. A trail of comments is maintained throughout the life cycle of the task. When adding attachments, you can use an absolute path name or browse for a file or provide a URL.

**History Section**

Figure 16–9 shows the short history for a vacation approval task.

**Figure 16–9  History Section of the Task Details Page**

The short history for a task lists all versions created by the following tasks:

- Initiate task
- Reinitiate task
- Update outcome of task
- Completion of task
- Erroring of task
- Expiration of task
- Withdrawal of task
- Alerting of task to the error assignee

You can include the following actions in the short history list by modifying the shortHistoryActions element in

```
SOA_Oraclle_Home/bpel/system/services/config/wf_config.xml
```

- Acquire
- Adhoc route
Features of the Worklist Application

- Auto release of task
- Delegate
- Escalate
- Information request on task
- Information submit for task
- Override routing slip
- Update outcome and route
- Push back
- Reassign
- Release
- Renew
- Resume
- Skip current assignment
- Suspend
- Update

The full history lists all version changes in a task.

Routing
If there is no predetermined sequence of approvers or if the workflow was designed to permit ad hoc routing, then the task can be routed in an ad hoc fashion. For such tasks, a Route button appears on the Task Details page. From the Routing page, you can look up one or more users for routing. When you specify multiple assignees, you can choose whether the list of assignees is for simple (group assignment to all users), sequential, or parallel assignment. In the case of parallel assignment, you provide the percentage of votes required for approval.

Requesting More Information
From the Task Details page, you can request more information by using the Request Info button. The Reapproval Needed field appears if previous approvers must reapprove given the additional information, assuming that the process was designed to support reapproval. You can also add comments. After you have requested additional information, the task is assigned to the user from whom the additional information is needed. The user from whom additional information is requested uses Submit More Info to fulfill the request.

Reassignment
From the Task Details page, you can reassign a task using the Reassign button. As Figure 16-10 shows, you can either reassign (transfer) or delegate.
Features of the Worklist Application

Figure 16–10  Reassigning Tasks

- **Reassign (transfer task to another user or group)**—The task is moved from the assignee’s worklist to another user’s worklist. The newly assigned user then acts on the task, rather than the original user.

- **Delegate (allow specified user to act on my behalf)**—The task is delegated to another user, but it shows up in both the original user’s and the delegated user’s worklists. The delegated user can act on behalf of the original assignee.

Use the **Search** button to find assignees and the up and down arrows to select or deselect assignees. Wildcard search is supported.

A supervisor can always reassign tasks to any of his reportees. Users with the BPMWorkflowReassign role can assign tasks to any users in the organization.
Parallel Tasks
Parallel tasks are created when a parallel flow pattern is specified for scenarios such as voting. In this pattern, the parallel tasks have a common parent. The parent task is visible to a user only if the user is an assignee or an owner or creator of the task. The parallel tasks themselves (referred to as subtasks) are visible to whomever the task is assigned, just like any other task. It is possible to view the subtasks from a parent task. In such a scenario, the Task Details page of the parent task contains a View SubTasks button. The SubTasks page lists the corresponding parallel tasks. In a voting scenario, if any of the assignees updates the payload or comments or attachments, the changes are visible only to the assignee of that task. A user who can view the parent task (such as the final reviewer of a parallel flow pattern), can drill down to the subtasks and view the updates made to the subtasks by the participants in the parallel flow.

Determining Action Permissions
A user can view a task when associated with the task as one of the following: current assignee (directly or by group membership), current assignee’s manager, creator, owner, or a previous actor.
A user’s profile determines his group memberships and roles. The roles determine a user’s privileges. Apart from the privileges, the exact set of actions a user can perform is also determined by the state of the task, the custom actions, and restricted actions defined for the task flow at design time.
The following algorithm is used to determine the actions a user can perform on a task:
1. Get the list of actions a user can perform based on the privileges granted to him.
2. Get the list of actions that can be performed in the current state of the task.
3. Create a combined list of actions that appear on the preceding lists.
4. Remove any action on the combined list that is specified as a restricted action on the task.
The resulting list of actions is displayed in the listing page and the Task Details page for the user. When a user requests a specific action, such as claim, suspend, or reassign, the workflow service ensures that the requested action is contained in the list determined by the preceding algorithm.
Step 2 in the preceding algorithm deals with many cases. If a task is in a final, completed state (after all approvals in a sequential flow), an expired state, a withdrawn state, or an errored state, then no further update actions are permitted. In any of the these states, the task, task history, and subtasks (parent task in parallel flow) can be viewed. If a task is suspended, then it can only be resumed or withdrawn. A task that is assigned to a group must be claimed before any actions can be performed on it.
See “Identity Service” on page 15-102 for information about the identity service and how privileges can be assigned to users.

Using Advanced Search
If you click the Advanced Search link, the page shown in Figure 16-11 is displayed.
When you search on a task type, the Select Workflow Task Type page is displayed. From this page, you select a task type and are returned to the Advanced Search page. As Figure 16–12 shows, you can filter the search by adding conditions.
Conditions can be AND operations (the All of the following option) or OR operations (the Any of the following option). Each filter specifies a combination of attribute, operator, and value. The operator and value are tied to the type of the attribute and change based on the attribute chosen. For example, for identity fields such as Created By or Updated By, a flashlight icon appears so that you can search for names using the identity browser. For date fields, a calendar icon appears so that you can pick a date.

**Viewing a Bar Chart of Task Status**

When you click the bar chart icon, a bar chart of the tasks is displayed, as shown in Figure 16–13.
Features of the Worklist Application

Figure 16–13  My Tasks Page with Chart Displayed

The bar chart shows the tasks broken down by status, with a count of how many tasks in each status category.

Using Work Queues

The Work Queues pane, shown in Figure 16–14, is displayed by default. (Use the work queues icon to reopen a closed pane.)
The Work Queues pane displays the following:

- **Inbox**—Shows all tasks that qualify for the user-chosen filter. The default shows all tasks, including high priority tasks, tasks due soon, new tasks, and so on.
- **My Work Queues**—Shows standard work queues, and custom work queues that users have defined based on specific search criteria.
- **Proxy Work Queues**—Shows queues to which a user has granted access to other users. Other users can act on those tasks on behalf of the user who granted access.

**Setting Preferences**

From the Preferences link, the following kinds of preferences are available:

- Vacation Preferences
- My Rules
- Group Rules
- Custom Views
- Display Preferences

**Vacation Preferences**

Use the vacation preferences to make yourself unavailable for task assignments. As Figure 16–15 shows, you specify a vacation date range, and optionally create a rule. Based on the rules you specify, tasks can be approved automatically or reassigned to someone else, for example.
As Figure 16–16 shows, when creating a rule, you can specify which task the rule applies to, add conditions, and delegate or reassign the task to another user or a group.
Features of the Worklist Application

- **Conditions on the rule**—These are filters that further define the rule, such as specifying that a rule acts on priority 1 tasks only, or that a rule acts on tasks created by a specific user. The conditions can be based on standard task attributes as well as any flex fields that have been mapped for the specific tasks. See "Using the Administration Functions" on page 16-27 for information about mapping flex fields.

Figure 16–17 shows how you add conditions to a rule.

**Figure 16–17  Adding Conditions on a Rule**

- **Actions**
  - **Reassign to**—You can reassign tasks to subordinates or groups you manage. If you have been granted the BPMWorkflowReassign role, then you can reassign tasks to any user or group.
  - **Delegate to**—You can delegate to any user or group.
  - **Set outcome to**—You can specify an automatic outcome if the workflow task was designed for those outcomes, for example, accepting or rejecting the task. The rule must be for a specific task type. If a rule is for all task types, then this option is not displayed.
  - **Take no action**—Use this action to prevent other more general rules from applying. For example, if you want to reassign all your tasks to another user while you are on vacation, with the exception of loan requests, for which you want no action taken, then create two rules. The first rule specifies that no action is taken for loan requests; the second rule specifies that all tasks are reassigned to another user. The first rule will prevent reassignment for loan requests.

Figure 16–18 shows the Rules List page. Rules are executed in the order in which they are listed. Use the Move Up and Move Down buttons to reorder rules. If a rule meets its filter conditions, then it is executed and no other rules are evaluated. For your rule to execute, you must be the only user assigned to that task. If the task is assigned to multiple users (including you), the rule does not execute.
**Features of the Worklist Application**

**Figure 16–18 Setting User Rules in Worklist Preferences**

Figure 16–18 also shows the following:

- You can create, delete, and edit rules (click the rule name).
- A rule is active (see the Active column in Figure 16–18) if the date range you specified when you created the rule is current.

**Group Rules**

Use a group rule to specify how a workflow rule applies to members of a group. Examples of group rules include:

- Assigning tasks from a particular customer to a member of the group
- Ensuring an even distribution of task assignments to members of a group by using round-robin assignment
- Ensuring that high-priority tasks are routed to the least busy member of a group

Creating a group rule is similar to creating other rules (see Figure 16–16, "Creating a Rule in the Worklist Application"); only some of the actions are different. For group rules, you can specify the following actions:

- **Reassign via**—You can specify a criterion to determine which member of the group gets the assignment. This dynamic assignment criterion can include round-robin assignment, assignment to the least busy group member, or assignment to the most productive group member. You can also add your custom functions for allocating tasks to users in a group. See the following for more information:
  - "Runtime Config Service" on page 15-111 for more information about dynamic assignment
  - "Implementing a Dynamic Assignment Function" on page 15-114 for more information about custom functions

- **Reassign to**—As with user rules, you can reassign tasks to subordinates or groups you directly manage. If you have been granted the BPMWorkflowReassign role, then you can reassign tasks to any user or group (outside your management hierarchy).

- **Take no action**—As with user rules, you can create a rule with a condition that prevents a more generic rule from being executed.
Feature of the Worklist Application

The group Rules List page is similar to the user Rules List page, with the addition of a list of the groups that you (as the logged-in user) manage. You can select from this list to specify the group for which you are creating a rule.

Custom Views

Use a custom view to customize your task list display. Examples of custom displays include:

- Ordering the task list in a particular way
- Displaying only those tasks that meet a particular condition
- Displaying specific attributes (columns) in your task list

You can also grant other users access to your views.

Figure 16–19 shows the Custom Views page.

The following functionality is available:

- You can create, edit, copy, and delete views, and choose to make the view visible or not in the My Views section of the Work Queues pane.
- For each view in the Granted Views list, you can choose to make the view visible or not in the Delegated Views section of the Work Queues pane on the Task List page.
- Details are available for granted views. You can rename a view granted to you.

Figure 16–20 shows the Create Custom View page.
You can specify the following when creating a custom view:

- **General**—You must specify a name for your view.
- **Columns**—You can specify which columns you want to display in your task list. The columns in the views can be standard task attributes as well as any flex fields that have been mapped for the specific task type. See "Using the Administration Functions" on page 16-27 for information about mapping flex fields.

  The default columns are the same as the columns in your inbox. You can also choose to show tasks actions in your task list and select ascending or descending order for a single column.

- **Filter**—You can specify which task categories you want to display, for example, My, Group, My & Group, and so on. You can also add conditions, for example, a condition that displays a task only when the title contains the words loan request.

- **Sharing**—You can grant access to this view to another user; for example, if jstein grants access to a My & Group category of tasks to jcooper, then jcooper will see jstein’s tasks and group tasks. Sharing a view with another user is similar to delegating all tasks that correspond to that view to the other user; that is, the other user can act on your behalf. Shared views are visible in the Proxy Work Queues section of the worklist (shown in Figure 16–14, "Work Queues Pane").

**Display Preferences**

Use display preferences to customize how tasks are displayed in your worklist. As Figure 16–21 shows, you can use the following options to customize the display:

- **Maximum number of tasks per page**
- **Page height in pixels**
- **Default ordering of tasks**
- **Show the following columns in the inbox view**
- **Show task actions in task list**
Using the Administration Functions

Administrators are users who have been granted the BPMWorkflowAdmin role. Administrators see the following tabs on the Worklist Application home page:

- Manage Rules
- Flex Field Mappings
- Application Customization

Manage Rules

An administrator uses the Manage Rules tab, shown in Figure 16–22, to view or edit the rules for any user or group.
Features of the Worklist Application

Figure 16–22  The Manage Rules Tab

This tab is useful if an administrator is needed to fix a problem with a rule. Also, for a user who no longer works for the company, administrators can set up a rule for that user so that all tasks assigned to the user are automatically assigned to another user or group.

Flex Field Mappings
An administrator uses the Flex Field Mappings tab, shown in Figure 16–23, to promote data from the payload to inline attribute flex fields. By promoting data to flex fields, the data becomes searchable and can be displayed as columns in the Task Listing (home) page.

Figure 16–23  The Flex Field Mappings Tab

Creating Labels  To create a flex field mapping, an administrator first defines a semantic label, which provides a more meaningful display name for the flex field attribute. Click Create Label to use the Create Payload Mapping Label interface, as shown in Figure 16–24.
As the figure shows, the label *amount* is mapped to the flex field `NumberAttribute1`. The payload attribute is also mapped to the label. In this example, the `Number` attribute type is associated with the *amount* label. The end result is that the value of the `Number` attribute is stored in the `NumberAttribute1` column, and *amount* is the column label for that value as displayed in the user’s task list. Labels can be reused for different task types. You can delete a label only if it is not used in any mappings.

A mapped payload attribute can also be displayed as a column in a custom view, and used as a filter condition in both custom views and workflow rules. The display name of the payload attribute is the attribute label that is selected when doing the mapping.

**Browsing All Mappings** When this option is selected, all flex field mappings defined for all task types are displayed, as shown in Figure 16–25.

To display all the payload attributes mapped to a particular label, click the respective row in the label table.
Editing Mappings by Task Type  When this option is selected, administrators can view or edit flex field mappings for a particular task type.

To edit mappings by task type:

1. Select Edit mappings by task type and click the flashlight icon.
2. Select a task type and click Select, as shown in Figure 16–26.

Figure 16–26  Selecting a Task Type

3. With the task type displayed in the Edit mappings by task type field, click Go.

All current mappings for the task type are displayed, as shown in Figure 16–27.
Features of the Worklist Application

4. Select a mapping label and click Select. Figure 16–28 shows a completed mapping.

If you want to create a new label, click Create Label and provide a label name, as shown in Figure 16–29. Note that the data type will be restricted based on the data type of the payload attribute.
5. To add a new mapping, click **Add Row** (if needed) and select a payload attribute from the list.

6. Click the flashlight icon and select a label.

**Restrictions**  
Note the following restrictions:

- Only simple type payload attributes can be mapped. Mapping specific simple types within a complex type is not supported.
- A flex field (and thus a label) can be used only once per task type.
- Data type conversion is not supported for the number or date data types. For example, you may not map a payload attribute of type string to a label of type number.

**Application Customization**

An administrator uses the Application Customization tab, shown in Figure 16–30, to customize the appearance of the Worklist Application.
Features of the Worklist Application

Figure 16–30  The Application Customization Tab

Values can be specified for the following parameters:

- **Login page realm label**—If the identity service is configured with multiple realms, then the Worklist Application login page displays a list of realm names. `LABEL_LOGIN_REALM` specifies the resource bundle key used to look up the label to display these realms. The term `realm` can be changed to fit the user community—terms such as `country`, `company`, `division`, or `department` may be more appropriate. Administrators can customize the resource bundle, specify a resource key for this string, and then set this parameter to point to the resource key.

  See "Customizing the Login Page" on page 16-44 for information on customizing the image on the login page.

- **Branding image location**—This is the image displayed in the top left corner of every page of the Worklist Application. (The Oracle logo is the default.) Administrators can provide a `.gif`, `.png`, or `.jpg` file for the logo. This file must be in the `public_html` directory of the Worklist Application.

  See "Customizing Header Information" on page 16-44 for information about the header.

- **Application resource bundle classname**—A resource bundle provides the strings displayed in the Worklist Application. By default, this is the class at:

  `oracle.bpel.services.workflow.resource.WorkflowResourceBundle`

  Administrators can change the strings shown in the application by copying `WorkflowResourceBundle` and creating their own. This parameter allows administrators to specify the classpath to this custom resource bundle.

Creating Reports

The Worklist Application offers the following reports from the Reports link:

- **Unattended Tasks Report**
- **Tasks Priority Report**
- **Tasks Cycle Time Report**
- **Tasks Productivity Report**

To create a report:
1. Click the Reports link.
2. Click the name of the report you want.
3. Provide inputs to define the search parameters of the report.
See the following sections on each report type for information about input parameters.

4. Click Run.

As shown in Figure 16–31, report results (for all report types) are displayed in both a table format and a bar chart format. The input parameters used to run the report are displayed under Report Inputs, in the lower-left corner (may require scrolling to view).

Figure 16–31 Report Display—Table Format, Bar Chart Format, and Report Inputs

Unattended Tasks Report

This report provides an analysis of tasks assigned to users’ groups or reportees’ groups that have not yet been claimed (unattended tasks). Use the following input parameters to define the report:

- Assignee—The tasks analyzed are based on the category chosen as it applies to the user; that is, tasks assigned to the user’s groups, tasks assigned to the reportee’s groups, tasks where the user is a creator, and tasks where the user is an owner.
- Creation Date (range)
- Expiration Date (range)
- Task State
- Priority

See Table 16–2 on page 16-6 for descriptions of Creation (or Created) Date, Expiration Date, Task State (or Status), and Priority.

Figure 16–32 shows an example of an Unattended Tasks Report.
The report shows that the California group has 15 unattended tasks, the Supervisor group has 7 unattended tasks, and the LoanAgentGroup has 11 unattended tasks. The unattended (unclaimed) tasks in this report are all DocumentReview tasks. If more than one type of unattended task exists when a report is run, all task types are included in the report, and the various task types are differentiated by color.

**Tasks Priority Report**

This report provides an analysis of the number of tasks assigned to a user, reportees, or their groups, broken down by priority. Use the following input parameters to define the report:

- **Assignee**—Depending on the assignee that you choose, this includes tasks assigned to you (My), tasks assigned to you and groups that you belong to (My & Group), or tasks assigned to groups to which your reportees belong.
- **Creation Date (range)**
- **Ended Date (range)**—This is the end dates of the tasks to be included in the report.
- **Priority**

See Table 16-2 on page 16-6 for descriptions of Creation (or Created) Date and Priority. Figure 16-33 shows an example of a Tasks Priority Report.
The report shows that the California group, the Supervisor group, and the LoanAgentGroup each have 16 tasks of normal priority. The users rsteven and jcooper have 5 and 22 tasks, respectively, all normal priority. Priorities (highest, high, normal, low, lowest) are distinguished by different colors in the bar chart.

Tasks Cycle Time Report
This report provides an analysis of the time taken to complete tasks from creation to completion based on users’ groups or reportees’ groups. Use the following input parameters to define the report:

- Assignee—Depending on the assignee that you choose, this includes your tasks or tasks assigned to groups to which your reportees belong.
- Creation Date (range)
- Ended Date (range)—This is the end dates of the tasks to be included in the report.
- Priority

See Table 16–2 on page 16-6 for descriptions of Creation (or Created) Date and Priority. Figure 16–34 shows an example of a Tasks Cycle Time Report.
Features of the Worklist Application

Figure 16–34  Tasks Cycle Time Report

The report shows that it takes 1 hour and 6 minutes on average to complete DocumentReview tasks, and 1 hour and 28 minutes on average to complete VacationApproval tasks. The bar chart shows the average cycle time in milliseconds.

Tasks Productivity Report

This report provides an analysis of assigned tasks and completed tasks in a given time period for a user, reportees, or their groups. Use the following input parameters to define the report:

- Assignee—Depending on the assignee that you choose, this includes your tasks or tasks assigned to groups to which your reportees belong.
- Creation Date (range)—The default is one week.
- Task Type—Use the flashlight icon to select from a list of task titles. All versions of a task are listed on the Select Workflow Task Type page, as shown in Figure 16–35.
Features of the Worklist Application

Figure 16–35 Select Workflow Task Type

Figure 16–36 shows an example of a Tasks Productivity Report.

Figure 16–36 Tasks Productivity Report

<table>
<thead>
<tr>
<th>User/Group Name</th>
<th>Task State</th>
<th>Number of Tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>California</td>
<td>Assigned</td>
<td>15</td>
</tr>
<tr>
<td>LoanAgentGroup</td>
<td>Assigned</td>
<td>11</td>
</tr>
<tr>
<td>Supervisor</td>
<td>Assigned</td>
<td>7</td>
</tr>
<tr>
<td>Jcooper</td>
<td>Assigned</td>
<td>22</td>
</tr>
<tr>
<td>Jmount</td>
<td>Completed</td>
<td>2</td>
</tr>
<tr>
<td>Jthom</td>
<td>Completed</td>
<td>5</td>
</tr>
<tr>
<td>Jstein</td>
<td>Completed</td>
<td>11</td>
</tr>
</tbody>
</table>

The report shows the number of tasks assigned to the California, LoanAgentGroup, and Supervisor groups. For individual users, the report shows that jcooper has 22 assigned tasks. In addition to his assigned tasks, jcooper has completed 2 tasks. The report shows that mtwain and rsteven have completed 6 and 11 tasks respectively. In the bar chart, the two task states—assigned and completed—are differentiated by color.

16-38 Oracle BPEL Process Manager Developer's Guide
User and Group Information

In the banner area, the logged-in user’s name appears, as in Welcome jstein [jazn.com]. Click the user name to display information such as the user’s full name, telephone number, e-mail address, manager, reportees, groups to which the user belongs, and roles that have been granted, as shown in Figure 16–37.

Figure 16–37 User Information

The roles that have been granted control the actions that the user can perform in the application. The user can click the manager and reportee links to get user information by traveling up and down the management chain. Clicking a group displays the Group Info page for that group. The Group Info page displays the list of direct and indirect users (users contained in child groups of the current group).

Accessing the Worklist Application in Local Languages

The identity service determines a user’s preferred language and time zone. This information is extracted from either the JAZN file-based community or from an external directory service such as Oracle Internet Directory. If no preference information is available, then the user’s preferred language and time zone are set to the system default (en_US and America/Los_Angeles, as shown in the following sample code).

Using the sample worklist configured with the user community in the JAZN XML file, you can set the user’s preferred language and time zone in the demo-users-properties.xml file as follows:

```xml
<timeZone>America/Los_Angeles</timeZone>
<languagePreference>en_US</languagePreference>
```
The demo-users-properties.xml file is found in
SOA_Oracle_Home\bpel\system\services\config

The Worklist Application supports the locales shown in Table 16–4.

Table 16–4 Languages and Java Locales Supported by the Worklist Application

<table>
<thead>
<tr>
<th>Language</th>
<th>Java Locale</th>
</tr>
</thead>
<tbody>
<tr>
<td>English</td>
<td>(en)</td>
</tr>
<tr>
<td>English (United States)</td>
<td>(en_US)</td>
</tr>
<tr>
<td>German</td>
<td>(de)</td>
</tr>
<tr>
<td>Spanish (International)</td>
<td>(es)</td>
</tr>
<tr>
<td>Spanish (Spain)</td>
<td>(es_ES)</td>
</tr>
<tr>
<td>French</td>
<td>(fr)</td>
</tr>
<tr>
<td>French (Canada)</td>
<td>(fr_CA)</td>
</tr>
<tr>
<td>Italian</td>
<td>(it)</td>
</tr>
<tr>
<td>Japanese</td>
<td>(ja)</td>
</tr>
<tr>
<td>Korean</td>
<td>(ko)</td>
</tr>
<tr>
<td>Portuguese</td>
<td>(pt)</td>
</tr>
<tr>
<td>Portuguese (Brazil)</td>
<td>(pt_BR)</td>
</tr>
<tr>
<td>Chinese (Simplified)</td>
<td>(zh_CN)</td>
</tr>
<tr>
<td>Chinese (Traditional)</td>
<td>(zh_TW)</td>
</tr>
</tbody>
</table>

If an LDAP-based provider such as OID is used, then language settings are changed in
the OID community.

When a user opens a browser and logs in to the Worklist Application, the worklist
screens are rendered in the browser’s locale and time zone. Most strings in the
Worklist Application come from the worklist application bundle. By default, this is the
class
oracle.bpel.services.workflow.resource.WorkflowResourceBundle

However, this can be changed to a custom resource bundle by setting the appropriate
application preference. See “Using the Administration Functions” on page 16-27 for
more information.

For task attribute names, flex field attribute labels, and dynamic assignment function
names, the strings come from configuring the resource property file
WorkflowLabels.properties. This file exists in the wfresource subdirectory of
the services config directory. See Chapter 15, "Oracle BPEL Process Manager Workflow
Services" for information on adding entries to this file for dynamic assignment
functions and attribute labels.

For custom actions and task titles, the display names come from the message bundle
specified in the task configuration file. If no message bundle is specified, then the
values specified at design time are used. See Chapter 15, "Oracle BPEL Process
Manager Workflow Services" for information on how to specify message bundles so
that custom actions and task titles are displayed in the preferred language.
Customizing the Worklist Application

The sample Worklist Application described in this chapter is fully functional. Use it as a starting point to create a customized Worklist Application to suit your specific needs. This section discusses the architecture of the Worklist Application and provides specific details for customizing it.

The Worklist Application is available in the samples directory at

```
SOA_Oracle_Home\bpel\samples\bx\worklistapp
```

Worklist Application Architecture

The Worklist Application follows the standard model-view-controller approach, as shown in Figure 16–38.

Figure 16–38 Worklist Application Architecture

A request coming from the browser is handled by a servlet. The servlet validates the request and calls the appropriate workflow service client API to query or update data. The worklist client APIs support a variety of different protocols (local and remote EJBs, direct java invocation, SOAP) for invoking the underlying workflow service. The clients send the API request to the workflow services, using the appropriate protocol. After the API call, the servlet stores the data required for rendering the next page in the session. The JSP picks up the data from the session, renders the data, and removes it from the session. Hence the servlets and the JSPs have different functions. The servlets are responsible for making the back-end API calls and the JSPs are responsible for formatting the data.

The Worklist Application servlets are at

```
SOA_Oracle_Home\bpel\samples\bx\worklistapp\src\worklistapp\servlets
```

All servlets extend the class `worklistapp.servlets.BaseServlet`. This class implements common functionality required by all servlets, such as authentication.

The JSPs are at

```
SOA_Oracle_Home\bpel\samples\bx\worklistapp\public_html
```

The workflow client API is a public interface made available by the workflow services. The interface is at

```
oracle.bpel.services.workflow.client.IWorkflowServiceClient
```

An instance of the API interface can be obtained by invoking the `getWorkflowServiceClient` method on

```
oracle.bpel.services.workflow.client.WorkflowServiceClientFactory
```
See Chapter 15, "Oracle BPEL Process Manager Workflow Services" for more information.

A typical page flow sequence is shown in Figure 16–39.

Figure 16–39 A Typical Page Flow Sequence

This sequence encompasses logging in to the application to view the details of a task. The first time a user enters the login URL, the login servlet redirects the page to the login JSP that is sent to the browser. The user enters a username and password and the login servlet calls the authenticate method on the task query service. If successful, it redirects to the TaskList servlet URL. The browser’s request then goes to the TaskList servlet that calls the queryTasks method on the task query service for getting the tasks that the user should see. Then it redirects the page to the TaskList JSP that is sent to the browser. When a user clicks a task link, the request is handled by the TaskDetails servlet. This calls the getTaskDetailsById method on the task query service and redirects the page to the TaskDetails JSP that is sent to the browser. Page flows for other functionality, such as updating the payload, adding an attachment, reassigning a task, viewing history, and updating user preferences, are similar.

The separation of responsibility—between servlets that handle API calls and processing, and JSPs that handle formatting of the data—facilitates customizing the application. The page flow requirements for many customer requirements are probably similar to the page flow for the sample Worklist Application. Therefore, it
may be sufficient to modify the JSPs (and the Java class `HTMLFormatter.java` used for formatting HTML data).

Table 16–5 lists the Worklist Application JSPs.

<table>
<thead>
<tr>
<th>JSP</th>
<th>Servlet</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>AdminPrefs.jsp</td>
<td>Admin</td>
<td>Application customization preferences</td>
</tr>
<tr>
<td>AdvancedSearch.jsp</td>
<td>TaskList</td>
<td>Advanced query for tasklist</td>
</tr>
<tr>
<td>Branding.jsp</td>
<td>--</td>
<td>Branding information displayed in the top-left corner of every page</td>
</tr>
<tr>
<td>ColumnSelectIncludes.jsp</td>
<td>--</td>
<td>Control that allows users to select a list of columns. Used in DisplayPrefs.jsp and ViewEdit.jsp</td>
</tr>
<tr>
<td>DisplayPrefs.jsp</td>
<td>Preferences</td>
<td>User display preferences</td>
</tr>
<tr>
<td>Error.jsp</td>
<td>--</td>
<td>All servlets redirect to this page when the exception is caught</td>
</tr>
<tr>
<td>FilterForm.jsp</td>
<td>--</td>
<td>Control that allows users to define advanced task queries. Used in AdvancedSearch.jsp and ViewEdit.jsp</td>
</tr>
<tr>
<td>FilterIncludes.jsp</td>
<td>--</td>
<td>Control that allows users to define task filtering criteria, used in FilterForm.jsp and RuleEdit.jsp</td>
</tr>
<tr>
<td>Footer.jsp</td>
<td>--</td>
<td>Appears at the bottom of every page</td>
</tr>
<tr>
<td>GetHWTaskHistory.jsp</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>Header.jsp</td>
<td>--</td>
<td>Appears at the top of every page</td>
</tr>
<tr>
<td>HeaderIncludes.jsp</td>
<td>--</td>
<td>Used to include common Javascript function into the page headers</td>
</tr>
<tr>
<td>Home.jsp</td>
<td>Admin</td>
<td>Used as a container for the administrator pages</td>
</tr>
<tr>
<td>IdentityBrowser.jsp</td>
<td>IdentityBrowserPopup</td>
<td>Control that allows users to select users and groups</td>
</tr>
<tr>
<td>IdentityBrowserPopup.jsp</td>
<td>IdentificationBrowserPopup</td>
<td>Pop-up window that includes the identity browser control</td>
</tr>
<tr>
<td>Login.jsp</td>
<td>Login</td>
<td>Application login page</td>
</tr>
<tr>
<td>PayloadMapping.jsp</td>
<td>Admin</td>
<td>Flex field payload mapping</td>
</tr>
<tr>
<td>PayloadMappingBrowser.jsp</td>
<td>PayloadMappingBrowser</td>
<td>Flex field payload mapping</td>
</tr>
<tr>
<td>PayloadMappingBrowserPopup.jsp</td>
<td>PayloadMappingBrowserPopup</td>
<td>Flex field payload mapping</td>
</tr>
<tr>
<td>PayloadMappingEditor.jsp</td>
<td>--</td>
<td>Flex field payload mapping</td>
</tr>
<tr>
<td>PayloadMappingLabelPopup.jsp</td>
<td>--</td>
<td>Flex field payload mapping</td>
</tr>
<tr>
<td>PopUpHeader.jsp</td>
<td>--</td>
<td>Header displayed in pop-up windows</td>
</tr>
<tr>
<td>Preferences.jsp</td>
<td>Preferences</td>
<td>Used as a container for the user preferences pages</td>
</tr>
<tr>
<td>ReportChart.jsp</td>
<td>Reports</td>
<td>Task reporting</td>
</tr>
<tr>
<td>ReportEdit.jsp</td>
<td>Reports</td>
<td>Task reporting</td>
</tr>
<tr>
<td>ReportInput.jsp</td>
<td>Reports</td>
<td>Task reporting</td>
</tr>
</tbody>
</table>
The following sections discuss how to customize some commonly used pages.

### Customizing the Login Page

You can customize the image on the login page (the default is an image of people). In `Login.jsp`, replace the portion of the image tag shown in bold (`people.jpg`) with your own image:

```html
<IMG HEIGHT="55" SRC="img/people.jpg">
```

See “Application Customization” on page 16-32 for information on customizing the login page realm label.

### Customizing Header Information

The header section appears on every page above the bread crumb navigation. You can customize the header by modifying the `Header.jsp` file. The logo and the name of the application in the left corner are contained in the `Branding.jsp` file that is included in the header.

See “Application Customization” on page 16-32 for information on changing the branding image.

---

**Table 16–5 (Cont.) Worklist Application JSPs**

<table>
<thead>
<tr>
<th>JSP</th>
<th>Servlet</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ReportOutput.jsp</code></td>
<td>Reports</td>
<td>Task reporting</td>
</tr>
<tr>
<td><code>Reports.jsp</code></td>
<td>Reports</td>
<td>Container page for the task reporting pages</td>
</tr>
<tr>
<td><code>RequestInfo.jsp</code></td>
<td>RequestInfo</td>
<td>Task request info requests</td>
</tr>
<tr>
<td><code>RuleCreate.jsp</code></td>
<td>Preferences</td>
<td>Create a new workflow rule</td>
</tr>
<tr>
<td><code>RuleEdit.jsp</code></td>
<td>Preferences</td>
<td>Edit workflow rule details</td>
</tr>
<tr>
<td><code>RuleList.jsp</code></td>
<td>Preferences</td>
<td>Listing of workflow rules</td>
</tr>
<tr>
<td><code>SubTasks.jsp</code></td>
<td>SubTasks</td>
<td>View task subtasks</td>
</tr>
<tr>
<td><code>TaskAssignes.jsp</code></td>
<td>TaskAssignee</td>
<td>Handle task reassignment</td>
</tr>
<tr>
<td><code>TaskDetails.jsp</code></td>
<td>TaskDetails</td>
<td>Display task details</td>
</tr>
<tr>
<td><code>TaskHistory.jsp</code></td>
<td>TaskHistory</td>
<td>Display task history</td>
</tr>
<tr>
<td><code>TaskList.jsp</code></td>
<td>TaskList</td>
<td>Main application page. Displays lists of tasks</td>
</tr>
<tr>
<td><code>TaskRouting.jsp</code></td>
<td>TaskRouting</td>
<td>Handle updates to task routing</td>
</tr>
<tr>
<td><code>TaskTypeDetails.jsp</code></td>
<td>TaskType</td>
<td>Display details for workflow tasktype</td>
</tr>
<tr>
<td><code>TaskTypeList.jsp</code></td>
<td>TaskType</td>
<td>Display a list of task types in a pop-up window</td>
</tr>
<tr>
<td><code>UserInfo.jsp</code></td>
<td>UserInfo</td>
<td>Display user information</td>
</tr>
<tr>
<td><code>UserInfoContent.jsp</code></td>
<td>UserInfo</td>
<td>Display user information</td>
</tr>
<tr>
<td><code>Vacation.jsp</code></td>
<td>Preferences</td>
<td>User vacation preference</td>
</tr>
<tr>
<td><code>ViewDetails.jsp</code></td>
<td>Preferences</td>
<td>Details for delegated user task views</td>
</tr>
<tr>
<td><code>ViewEdit.jsp</code></td>
<td>Preferences</td>
<td>Edit details for owned user task views</td>
</tr>
<tr>
<td><code>ViewList.jsp</code></td>
<td>Preferences</td>
<td>Listing of user task views</td>
</tr>
</tbody>
</table>

---
The upper-right area contains HTML controls for filters and search criteria for retrieving tasks. The filters can be customized to include only those choices that are relevant to the application.

Customizing the Task Details Page

The Task Details page is used to examine the contents of the task and view or update the payload. The layout of the details page consists of the actions and buttons at the top, a header section, the payload section, and the footer section consisting of optional contents such as comments and attachments. The information displayed on this form is typically defined by the task definition for the task being displayed (and the format is controlled by the workflow task designer). You can customize the Task Details page by modifying the TaskDetails.jsp file. See "Generating a Custom Task Display Form" on page 15-74 for information on how to customize this file.

Changing the Client-Service Binding for the Worklist Application

The workflow services client interfaces can use a number of protocols to communicate with the workflow services. The client implementations encapsulate all the communication details, and users of the client interfaces do not need to be concerned with the details.

The Worklist Application is deployed in the same container as the workflow services, by default, and the application uses the Java client.

To switch the client type used by the Worklist Application, modify the init method in BaseServlet.java as follows:

```java
public void init(ServletConfig config) throws ServletException {
    super.init(config);
    try {
        wfSvcClient = WorkflowServiceClientFactory.getWorkflowServiceClient(
            WorkflowServiceClientFactory.JAVA_CLIENT);
    }
    catch (Exception e) {
        wlSvcError = getStackTraceString(e);
    }
}
```

Also, change WorkflowServiceClientFactory.JAVA_CLIENT to one of the following:

- WorkflowServiceClientFactory.SOAP_CLIENT—to use the SOAP-based Web services interface
- WorkflowServiceClientFactory.LOCAL_CLIENT—to use the local EJB interface
- WorkflowServiceClientFactory.REMOTE_CLIENT—to use the remote EJB interface

In addition, ensure that the wf_client_config.xml file is correctly set up for the client type that you select.
Deploying the Custom Worklist Application

The top-level directory of the sample Worklist Application contains an ant script, build.xml, that can be used to build and deploy the Worklist Application. This ant script makes use of the properties file orabpel.properties that exists in the same directory. The instructions in this section also provide fixes to some incorrect files in the sample Worklist Application source files configuration.

This section contains the following topics:

- Task 1: Changing the Application Configuration
- Task 2: Changing the Build File
- Task 3: Reviewing the File Changes
- Task 4: Building and Deploying the Application

See Also: "Enabling the Worklist Application for Single Sign-On" on page 16-50 after performing these tasks if you want to secure the Worklist Application to be Java single sign-on (JSSO)-enabled

Task 1: Changing the Application Configuration

The sample web.xml and worklist-taglib.tld files are not currently in sync with the ones in the deployed Worklist Application and must be overwritten.

1. Copy the web.xml file from:
   SOA_Oracle_Home\j2ee\OC4J_instance_name\applications\hw_services\worklistapp\WEB-INF\n
to:
   SOA_Oracle_Home\bpel\samples\hw\worklistapp\config\n
2. Copy the worklist-taglib.tld file from:
   SOA_Oracle_Home\j2ee\OC4J_instance_name\applications\hw_services\worklistapp\WEB-INF\n
to:
   SOA_Oracle_Home\bpel\samples\hw\worklistapp\public_html\WEB-INF\n
3. Ensure that the EJB Server URL in the following file is the same as the value you specify for the oc4j.ormi.url property in Step 5 of "Task 2: Changing the Build File" on page 16-47.
   SOA_Oracle_Home\bpel\system\services\config\wf_client_config.xml

Task 2: Changing the Build File

You must update the SOA_Oracle_Home\bpel\samples\hw\worklistapp\build.xml file for your process.

1. Update the path element that specifies the classpath to contain the following:
   <path id="classpath">
   <pathelement location="${orabpel.home}/system/services/lib/bpm-services.jar" />
   <pathelement location="${orabpel.home}/lib/orabpel.jar" />
   <pathelement location="${orabpel.home}/lib/bicmn.jar" />
   <pathelement location="${orabpel.home}/lib/bipres.jar" />
   </path>

See Also: "Enabling the Worklist Application for Single Sign-On" on page 16-50 after performing these tasks if you want to secure the Worklist Application to be Java single sign-on (JSSO)-enabled
Customizing the Worklist Application

2. Update the deploy target deploy.oc4j to include the following:

```xml
<target name="deploy.oc4j" depends="validate.properties,worklist.ear">
  <!-- Deploy application ear -->
  <java jar="${j2ee.home}/admin_client.jar" dir="${j2ee.home}" fork="true">
    <arg line="${oc4j.ormi.url} ${oc4j.admin.username} ${oc4j.admin.password} -deploy -file
      ${worklist.classes.dir}/customworklist.ear -deploymentName
      customapp -parent orabpel" />
  </java>
  <java jar="${j2ee.home}/admin_client.jar" dir="${j2ee.home}" fork="true">
    <arg line="${oc4j.ormi.url} ${oc4j.admin.username} ${oc4j.admin.password} -bindAllWebApps -appName customapp" />
  </java>
</target>
```

3. Update the target worklistapp.ear to include the following:

```xml
<target name="worklist.ear" depends="worklist.war">
  <ear compress="true" earfile="${worklist.classes.dir}/customworklist.ear"
    appxml="${worklist.config.dir}/application.xml">
    <fileset dir="${worklist.classes.dir}">
      <include name="**/*.war"/>
    </fileset>
    <fileset dir="${worklist.config.dir}">
      <include name="**/META-INF/orion-application.xml"/>
    </fileset>
  </ear>
</target>
```

4. Open the SOA_Oracle_Home\bpel\samples\hw\worklistapp\orabpel.properties file.

5. Change OC4J-related properties to the following:

```
#Standalone OC4J related properties
#oc4j.ormi.url=deployer:oc4j:hostname:ormiport
oc4j.ormi.url=deployer:oc4j:opmn://host_name:OPMN_request_port/home
oc4j.admin.username=oc4jadmin
oc4j.admin.password=password
```

Task 2: Reviewing the File Changes

Verify that you have correctly made the changes described in "Task 1: Changing the Application Configuration" through "Task 2: Changing the Build File" before attempting to build and deploy the application.

1. Verify that you correctly updated the following files in the SOA_Oracle_Home\bpel\samples\hw\worklistapp directory:
   - config\web.xml
   - build.xml
Customizing the Worklist Application

- orabpel.properties

2. Verify that you correctly updated the SOA_Oracle_Home\bpel\samples\hw\worklistapp\public_html\WEB-INF\worklist-taglib.tld file.

Task 4: Building and Deploying the Application

1. Ensure all the properties in orabpel.properties have been updated to reflect your environment.
2. Build and deploy the customized Worklist Application from the command line:
   ```
   ant deploy.oc4j
   ```
3. Access the customized Worklist Application at the following URL:
   ```
   http://host:port/integration/customapp/
   ```
4. Log in to the Worklist Application.
   The task list page appears.

See Also: "Enabling the Worklist Application for Single Sign-On" on page 16-50 if you now want to secure the Worklist Application to be Java single sign-on (JSSO)-enabled

Customizing the Worklist Application Using Preferences

The Worklist Application offers a number of ways to customize its look-and-feel without editing the JSP code or changing the application servlets.

Every worklist user is able to customize the columns displayed in his inbox, the size of the worklist page, and how many tasks to display at a time. See "Setting Preferences" on page 16-21 for more information.

Worklist administrators can also change a number of preferences that influence the appearance of the Worklist Application for all users, such as the branding image. Administrators can specify a different resource bundle and change the label for the list of realms on the login screen. See "Using the Administration Functions" on page 16-27 for more information.

Configuring Display Names for Task Attributes Using WorkflowLabels.properties

You can change the names used for various task attributes in the application by updating the following file (and its associated translations): SOA_Oracle_Home\bpel\system\services\config\wfresource\WorkflowLabels.properties

Note that this changes the labels returned by the task metadata service methods getTaskAttributes and getTaskAttributesForTaskDefinition. Any service clients that use these methods will be affected.

Controlling Access to Information and Actions for Different Users

The workflow service uses the identity service that supports the JAZN file-based community or LDAP communities such as Oracle Internet Directory. A static set of role-actions (privileges) has been defined and assigned to roles. Users then get those privileges by way of roles assigned to them. The most important of the role-actions currently defined include:
Customizing the Worklist Application

- CLAIM
- WITHDRAW
- ESCALATE
- RENEW
- RELEASE
- REQUEST_INFO
- SUBMIT_INFO
- CUSTOM
- ADMIN
- REASSIGN
- SUSPEND
- RESUME
- VIEW_TASK_HISTORY

The role-actions apply globally; that is, at the application level and not at the process level or instance level.

You can customize the Worklist Application so that the information viewed and the actions performed on a given page are altered for different sets of users. The first part consists of creating new roles and assigning them to the required users. Then, in the JSP, the identity service can be used to check if the user has the granted role and to determine which code path to take.

For example, you can create a new role called BPMProcessingManager in jazn-data.xml. This file is at

```
SOA_Oracle_Home/lpsl/system/appserver/oc4j/j2ee/home/config
```

The required users must be assigned this role, as shown in the following code example:

```
...  
   <role>
      <name>BPMProcessingManager</name>
      <members>
         <member>
            <type>user</type>
            <name>jstein</name>
         </member>
      </members>
   </role>
...  
```

If an LDAP-based service such as OID is used, then these roles must be created and granted to users in that service.

The JSP code can be customized using the identity service as follows.

```
import="oracle.tip.pc.services.common.ServiceFactory"
import="oracle.tip.pc.services.identity.*"

boolean canEditTaskHeaderPriority = false;
// get info from identity service
try
```
Customizing the Worklist Application

```java
{  
  BPMAuthorizationService authorizationService =  
      ServiceFactory.getAuthorizationServiceInstance(realm);  
  // lookup user based on worklist context user  
  BPMUser bpmUser = authorizationService.lookupUser(user);  
  
  // check for BPMProcessingManager role  
  if (bpmUser.isInRole("BPMProcessingManager"))  
    canEditTaskHeaderPriority = true;  
}

try {  
  catch (Exception e)  
  {  
    out.println("Could not get information from identity service");  
  }  

  // use the canEditTaskHeaderPriority flag to control HTML behavior  
  if (canEditTaskHeaderPriority)  
    // display the priority information & edit controls  
    else  
      // just display the priority information
```

Enabling the Worklist Application for Single Sign-On

The 10.1.3.1 Worklist Application by default uses a custom authentication mechanism through its own login page. The Worklist Application does not run under OC4J container security and is not Java single sign-on (JSSO)-enabled. The Worklist Application source files are located in the `SOA_Oracle_Home\bpel\samples\hw\worklistapp` directory. This section describes how to secure the Worklist Application with JSSO.

This section contains the following topics:

- Task 1: Changing the Servlet Code
- Task 2: Changing the Application Configuration
- Task 3: Reviewing the File Changes
- Task 4: Building and Deploying the Application

**Note:** Before performing the tasks in this section, ensure that you have completed the Worklist Application deployment tasks in “Deploying the Custom Worklist Application” on page 16-46.

**Task 1: Changing the Servlet Code**

1. Open the `SOA_Oracle_Home\bpel\samples\hw\worklistapp\src\worklistapp\servlets\BaseServlet.java` file.

2. Remove the following code fragment that begins on line 218 in the `validateSession()` method:

```
else  
  {  
    // forward request to login page, if user session is null  
    // (not if session store or wfCtx is null, as login servlet will set them)  
    if (userSession == null)  
      {  
        RequestDispatcher rd = getServletContext().getRequestDispatcher(  
            WorklistappConstants.PAGE_LOGIN_JSP);  
```
if (rd != null) {
    rd.forward(request, response);
    return false;
} }

3. Open the Login.java file in the same directory.

4. Replace the code up to the end of the try{} block in the handleRequest() method with the following:

```
String user = getParameter(request, WorklistappConstants.PARAM_LOGIN_USER);
String password = getParameter(request, WorklistappConstants.PARAM_LOGIN_PASSWORD);
String realm = getParameter(request, WorklistappConstants.PARAM_LOGIN_REALM);
String redirectURL = getParameter(request, WorklistappConstants.PARAM_REDIRECT_URL);
HttpSession userSession = request.getSession(true);
SessionStore sessStore = new SessionStore(userSession);
String remoteUser = request.getRemoteUser();
if ((user == null) && (password == null) && (remoteUser == null)) {
    pageRedirect(request, response, WorklistappConstants.PAGE_LOGIN_JSP);
    return;
}
try {
    IWorkflowContext wfCtx = null;
    if (user != null) {
        //Authenticate the supplied credentials
        wfCtx = wfSvcClient.getTaskQueryService().authenticate(user, password, realm, null);
    } else {
        //Create context using remoteUser in request (pre-authenticated request)
        wfCtx = wfSvcClient.getTaskQueryService().createContext(request);
    }
    initSessionAttributes(sessStore, wfCtx);
    initRequestStatus(sessStore);
    if (redirectURL != null)
        response.sendRedirect(redirectURL);
    else
        response.sendRedirect(WorklistappConstants.SERVLET_TASK_LIST);
} }
```

5. Open the Logout.java file in the same directory.

6. Add the following private method:

```
//When logging out, we need to know if oc4j SSO util class
//exists in classpath
private static boolean oc4jSSOExists = false;
static
```
Customizing the Worklist Application

[ //Try and load the class
try {
    Class.forName("oracle.security.jaxn.sso.util.JSSOUtil");

    //Class was found
    oc4jSSOExists = true;
} catch (ClassNotFoundException e) {
    oc4jSSOExists = false;
}
]

7. Replace the code inside the try{} block in the handleRequest() method with the following:

   // destroy context not needed anymore
   wfSvcClient.getTaskQueryService().destroyWorkflowContext(wfCtx);
   //If we're running in oc4j, ensure SSO knows we've logged out...
   if (oc4jSSOExists) {
       oracle.security.jaxn.sso.util.JSSOUtil.logout(response,
       request.getContextPath() + WorklistappConstants.SERVLET_LOGIN);
   }
   //Invalidate the session
   if (userSession != null) {
       userSession.invalidate();
   }
   //If we're not running in oc4j, handle the redirect to the login page ourselves
   if (!oc4jSSOExists) {
       response.sendRedirect(WorklistappConstants.SERVLET_LOGIN);
   }

Task 2: Changing the Application Configuration

1. Go to the SOA_Oracle_Home\bpel\samples\hw\worklistapp\config directory.
2. Create a directory named META-INF.
3. Go into the META-INF directory and create a file named orion-application.xml.
4. Enable deployment by adding the following syntax to orion-application.xml.

   <?xml version='1.0' encoding='windows-1252' ?>
   <orion-application>
       <jazn provider="XML">
           <jazn-web-app auth-method="CUSTOM_AUTH"/>
       </jazn>
       <security-role-mapping name="PUBLIC">
           <group name="{PUBLIC}"/>
       </security-role-mapping>
   </orion-application>

   The sample web.xml file is not currently in sync with the one in the deployed Worklist Application and must be overwritten.
5. Add the following code to web.xml in the config directory.

```xml
<login-config>
  <auth-method>BASIC</auth-method>
</login-config>

<security-role>
  <role-name>{{PUBLIC}}</role-name>
</security-role>

<security-constraint>
  <web-resource-collection>
    <web-resource-name>worklistpages</web-resource-name>
    <url-pattern>/*</url-pattern>
  </web-resource-collection>
  <auth-constraint>
    <role-name>{{PUBLIC}}</role-name>
  </auth-constraint>
</security-constraint>

6. If you are using multiple realms, the SOA_Oracle_Home\bpel\system\services\config\wf_client_config.xml file must contain the correct entry for the realm in the following element:

```xml
<portal>
  <realmMapping>jazn.com</realmMapping>
</portal>
```

Task 3: Reviewing the File Changes

Verify that you have correctly made the changes described in "Task 1: Changing the Servlet Code" through "Task 2: Changing the Application Configuration" before attempting to build and deploy the application.

1. Verify that you correctly updated the following files in the SOA_Oracle_Home\bpel\samples\hw\worklistapp directory:
   - src\worklistapp\servlets\BaseServlet.java
   - src\worklistapp\servlets\Login.java
   - src\worklistapp\servlets\Logout.java

2. Verify that you correctly updated the SOA_Oracle_Home\bpel\system\services\config\wf_client_config.xml file.

3. Verify that you correctly created the SOA_Oracle_Home\bpel\samples\hw\worklistapp\config\META-INF\orion-application.xml file.

Task 4: Building and Deploying the Application

1. Ensure all the properties in orabpel.properties have been updated to reflect your environment.

2. Build and deploy the customized Worklist Application from the command line:
   ```
   ant deploy.oc4j
   ```

3. Access the customized Worklist Application at the following URL:
   ```
   http://host:port/integration/customapp/
   ```
   You are prompted with the SSO login page.
4. Log in to the Worklist Application.
   After you are authenticated, you see the task list page.
5. Log out of the Worklist Application.
   The SSO login page again appears.

**Building Clients for Workflow Services**

You can build clients for workflow services using the APIs exposed by the workflow service. The APIs enable clients to communicate with the workflow service using local and remote EJBs, SOAP, and HTTP.

You can start with the sample Worklist Application to build your own application.

The typical sequence of calls when building a simple worklist application is as follows:

1. Get a handle to IWorklistServiceClient from WorkflowServiceClientFactory.
2. Get a handle to ITaskQueryService from IWorklistServiceClient.
3. Authenticate a user by passing a username and password to the authenticate method on ITaskQueryService. Get a handle to IWorkflowContext.
4. Query the list of tasks using ITaskQueryService.
5. Get a handle to ITaskService from IWorklistServiceClient.
6. Iterate over the list of tasks returned, performing actions on the tasks using ITaskService.

**Example 16–1** demonstrates how to build a client for workflow services. A list of all tasks assigned to jstein is queried. A task whose outcome has not been set is approved.

```java
try {
    //Create JAVA WorkflowServiceClient
    IWorkflowServiceClient wfSvcClient = WorkflowServiceClientFactory.getWorkflowServiceClient(WorkflowServiceClientFactory.JAVA_CLIENT);
}

//Get the task query service
ITaskQueryService querySvc = wfSvcClient.getTaskQueryService();

//Login as jstein
IWorkflowContext ctx = querySvc.authenticate("jstein", "welcome", null, null); //Use default realm

//Set up list of columns to query
List queryColumns = new ArrayList();
queryColumns.add("TASKID");
queryColumns.add("TASKNUMBER");
queryColumns.add("TITLE");
queryColumns.add("OUTCOME");

//Create a predicate to query tasks that have a null outcome
String outcome = null;
Predicate predicate = new Predicate(TableConstants.WFTASK_OUTCOME_COLUMN, Predicate.OP_EQ, outcome);
```

Building Clients for Workflow Services

You can build clients for workflow services using the APIs exposed by the workflow service. The APIs enable clients to communicate with the workflow service using local and remote EJBs, SOAP, and HTTP.

You can start with the sample Worklist Application to build your own application.

The typical sequence of calls when building a simple worklist application is as follows:

1. Get a handle to IWorklistServiceClient from WorkflowServiceClientFactory.
2. Get a handle to ITaskQueryService from IWorklistServiceClient.
3. Authenticate a user by passing a username and password to the authenticate method on ITaskQueryService. Get a handle to IWorkflowContext.
4. Query the list of tasks using ITaskQueryService.
5. Get a handle to ITaskService from IWorklistServiceClient.
6. Iterate over the list of tasks returned, performing actions on the tasks using ITaskService.

**Example 16–1** demonstrates how to build a client for workflow services. A list of all tasks assigned to jstein is queried. A task whose outcome has not been set is approved.

```java
try {
    //Create JAVA WorkflowServiceClient
    IWorkflowServiceClient wfSvcClient = WorkflowServiceClientFactory.getWorkflowServiceClient(WorkflowServiceClientFactory.JAVA_CLIENT);
}

//Get the task query service
ITaskQueryService querySvc = wfSvcClient.getTaskQueryService();

//Login as jstein
IWorkflowContext ctx = querySvc.authenticate("jstein", "welcome", null, null); //Use default realm

//Set up list of columns to query
List queryColumns = new ArrayList();
queryColumns.add("TASKID");
queryColumns.add("TASKNUMBER");
queryColumns.add("TITLE");
queryColumns.add("OUTCOME");

//Create a predicate to query tasks that have a null outcome
String outcome = null;
Predicate predicate = new Predicate(TableConstants.WFTASK_OUTCOME_COLUMN, Predicate.OP_EQ, outcome);
```
//Create an ordering to order tasks by task number
Ordering ordering = new Ordering(TableConstants.WFTASK_TASKNUMBER_COLUMN, true, false);

//Query a list of tasks assigned to jstein
List tasks = querySvc.queryTasks(ctx, queryColumns, null, null, predicate, Ordering, 0);

//Get the task service
ITaskService taskSvc = wfSvcClient.getTaskService();

//Loop over the tasks, outputting task information, and approving tasks
for(int i = 0; i < tasks.size(); i++) {
    Task task = (Task)tasks.get(i);
    int taskNumber = task.getSystemAttributes().getTaskNumber();
    String title = task.getTitle();
    String taskId = task.getSystemAttributes().getTaskId();
    taskSvc.updateTaskOutcome(ctx, taskId, "APPROVED");
    System.out.println("Task #" + taskNumber + " \(" + title + ") is APPROVED");
}

//Handle any exceptions raised here...
/system.out.println("Caught workflow exception: " + e.getMessage());
}

See Also:
- The following samples, which demonstrate how to write a custom UI for the Worklist Application:
  - SOA_Oracle_home\bpel\samples\utils\AsyncLoanService\StarLoanUI
  - SOA_Oracle_home\bpel\samples\demos\HelpDeskServiceRequest\HelpDeskUI
  - SOA_Oracle_home\bpel\samples\demos\ExpenseRequestApproval\ExpenseRequestUI
- SOA_Oracle_Home\bpel\docs\workflow\index.html for Javadoc that describes the workflow service APIs

Packages and Classes for Building Clients

Use the following packages and classes for building clients:
- oracle.bpel.services.workflow.metadata.config.model
  - The classes in this package contain the object model for the workflow configuration in the task definition file. The ObjectFactory class can be used to create objects.
- oracle.bpel.services.workflow.metadata.routingslip.model
The classes in this package contain the object model for the routing slip. The ObjectFactory class can be used to create objects.

- `oracle.bpel.services.workflow.metadata.taskdisplay.model`
  The classes in this package contain the object model for the task display. The ObjectFactory class can be used to create objects.

- `oracle.bpel.services.workflow.metadata.taskdefinition.model`
  The classes in this package contain the object model for the task definition file. The ObjectFactory class can be used to create objects.

- `oracle.bpel.services.workflow.client.IWorkflowServiceClient`
  Interface for the workflow service client.

- `oracle.bpel.services.workflow.client.WorkflowServiceClientFactory`
  The factory for creating the workflow service client.

- `oracle.bpel.services.workflow.metadata.ITaskMetadataService`
  The interface for task metadata service.

- `oracle.bpel.services.workflow.task.ITaskService`
  The interface for task service.

- `oracle.bpel.services.workflow.task.IRoutingSlipCallback`
  The interface for callback class to receive callbacks during task processing.

- `oracle.bpel.services.workflow.task.IAssignmentService`
  The interface for the assignment service.

**Workflow Service Client**

Any workflow application accesses the various workflow services through the workflow service client. The workflow service client code encapsulates all the logic required for communicating with the workflow services using different local and remote protocols. After the workflow application has an instance of the workflow service client, it does not need to consider how the client communicates with the workflow services.

The advantages of using the client are as follows:

- Hides the complexity of the underlying connection mechanisms such as SOAP/HTTP and EJB
- Facilitates changing from using one particular invocation mechanism to another, for example from SOAP/HTTP to remote EJB
- Helps to program with Java APIs for service input/outputs instead of XML inputs/outputs for SOAP/HTTP or Java WSIF invocation mechanism

The following class is used to create instances of the IWorkflowServiceClient interface:

- `oracle.bpel.services.workflow.client.WorkflowServiceClientFactory`

WorkflowServiceClientFactory has a single method, getWorkflowServiceClient, which takes a single parameter, the client type. The client type can be one of the following:
Building Clients for Workflow Services

- WorkflowServiceClientFactory.JAVA_CLIENT—The client uses Java to invoke the workflow services directly.
- WorkflowServiceClientFactory.LOCAL_CLIENT—The client uses a local EJB interface to invoke the workflow services.
- WorkflowServiceClientFactory.REMOTE_CLIENT—The client uses a remote EJB interface to invoke workflow services located remotely from the client.
- WorkflowServiceClientFactory.SOAP_CLIENT—The client uses SOAP to invoke Web service interfaces to the workflow services, located remotely from the client.

Through the factory, it is possible to get the client libraries for all the workflow services. Table 16–6 shows the clients available for each of the services.

<table>
<thead>
<tr>
<th>Service Name</th>
<th>Supports SOAP Web Services</th>
<th>Supports Remote EJB</th>
<th>Supports Local EJB</th>
<th>Supports Plain Java APIs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task Service</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Task Query Service</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Task Metadata Service</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Task Reports Service</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>User Metadata Service</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Runtime Config Service</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Identity Service:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BPM Authentication Service</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BPM Authorization Service</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The client classes use the configuration file `wf_client_config.xml` for the service end points. This file is at

`SOA_Oracle_Home/bpel/system/services/config`

In the client classpath, this file should be in the classpath directly, meaning the containing directory should be in the classpath. The `wf_client_config.xml` file contains:

- A section for EJB configuration

```xml
<ejb>
  <serverURL>ormi://localhost/hw_services</serverURL> <!-- for stand alone -->
  <!--serverURL>opmn:ormi://localhost:home/hw_services</serverURL--> <!-- for opmn managed instance -->
  <user>oc4jadmin</user>
  <password>welcome1</password>
  <initialContextFactory>oracle.j2ee.rmi.RMIInitialContextFactory</initialContextFactory/>
</ejb>
```

- A section for SOAP end points for each of the services

```xml
<taskService>
  ...
</taskService>
```
The IWorkflowServiceClient Interface

The IWorkflowServiceClient interface provides methods, summarized in Table 16–7, for obtaining handles to the various workflow services interfaces.

Classpaths for Java Clients

The following JAR files are necessary for the Java client classpath.

- $BPEL_HOME/bpel/lib/bpm-infra.jar
- $BPEL_HOME/bpel/lib/orabpel-common.jar
- $BPEL_HOME/bpel/lib/orabpel-thirdparty.jar
- $BPEL_HOME/bpel/lib/orabpel.jar
- $BPEL_HOME/bpel/system/appserver/cc4j/j2ee/home/jazncore.jar
- $BPEL_HOME/bpel/system/appserver/cc4j/j2ee/home/cc4jclient.jar
- $BPEL_HOME/bpel/system/appserver/cc4j/lib/xml.jar
- $BPEL_HOME/bpel/system/appserver/cc4j/lib/xmlparserv2.jar
- $BPEL_HOME/bpel/system/appserver/cc4j/webservices/lib/orasasaj.jar

<table>
<thead>
<tr>
<th>Method</th>
<th>Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>getTaskService</td>
<td>oracle.bpel.services.workflow.task.ITaskService</td>
</tr>
<tr>
<td>getTaskQueryService</td>
<td>oracle.bpel.services.workflow.query.ITaskQueryService</td>
</tr>
<tr>
<td>getTaskReportService</td>
<td>oracle.bpel.services.workflow.report.ITaskReportService</td>
</tr>
<tr>
<td>getTaskMetadataService</td>
<td>oracle.bpel.services.workflow.metadata.ITaskMetadataService</td>
</tr>
<tr>
<td>getUserMetadataService</td>
<td>oracle.bpel.services.workflow.user.IUserMetadataService</td>
</tr>
<tr>
<td>getRuntimeConfigService</td>
<td>oracle.bpel.services.workflow.runtimeconfig.IRuntimeConfigService</td>
</tr>
<tr>
<td>getAuthenticationService</td>
<td>oracle.tip.pc.services.identity.BPMAuthenticationService</td>
</tr>
<tr>
<td>getAuthorizationService</td>
<td>oracle.tip.pc.services.identity.BPMAuthorizationService</td>
</tr>
</tbody>
</table>
Building Clients for Workflow Services

- $BPEL_HOME/bpel/system/appserver/oc4j/webservices/lib/soap.jar
- $BPEL_HOME/bpel/system/services/config
- $BPEL_HOME/bpel/system/services/lib/bpm-services.jar
- $BPEL_HOME/bpel/system/services/schema
- wsclient_extended.zip

Note: The wsclient_extended.jar file is available as a separate download from the Oracle Technology Network at http://download.oracle.com/otn/java/oc4j/1013/wsclient_extended.zip

See the chapter “Web Service Client APIs and JARs” in the section “Simplifying the Classpath with wsclient_extended.jar” in Oracle Application Server Web Services Developer’s Guide 10g Release 3 (10.1.3), at http://www.oracle.com/technology/documentation

EJB References in Web Applications

If a Web application uses the workflow service local EJBs, then the client application must do the following:

- The application must be a child application of the hw_services application.
- The application must define the EJB local references in its web.xml file. The local references for each of the services are shown in Example 16–2 and Example 16–3.

Example 16–2 Task Service

```xml
<ejb-local-ref id="EjbRef_TaskServiceBean_Message">
  <ejb-ref-name>ejb/local/TaskServiceBean</ejb-ref-name>
  <ejb-ref-type>Session</ejb-ref-type>
  <local-home>oracle.bpel.services.workflow.task.ejb.TaskServiceLocalHome</local-home>
  <local>oracle.bpel.services.workflow.task.ejb.TaskServiceLocal</local>
  <ejb-link>TaskServiceBean</ejb-link>
</ejb-local-ref>
```

Example 16–3 Task Metadata Service

```xml
<ejb-local-ref id="EjbRef_TaskMetadataServiceBean_Message">
  <ejb-ref-name>ejb/local/TaskMetadataServiceBean</ejb-ref-name>
  <ejb-ref-type>Session</ejb-ref-type>
  <local-home>oracle.bpel.services.workflow.metadata.ejb.TaskMetadataServiceLocalHome</local-home>
  <local>oracle.bpel.services.workflow.metadata.ejb.TaskMetadataServiceLocal</local>
  <ejb-link>TaskMetadataServiceBean</ejb-link>
</ejb-local-ref>
```

Note: Only child applications can use local EJBs. This restricts standalone Java clients to using either remote EJBs or SOAP clients.

See Chapter 15, “Oracle BPEL Process Manager Workflow Services” for more information on the task query service, task report service, user metadata service, and runtime config service.
Initiating a Task

Tasks can be initiated programmatically, in which case the following task attributes must be set:

- taskDefinitionURI
- title
- payload
- priority

The following task attributes are optional, but are typically set by clients:

- creator
- ownerUser—Defaults to bpeladmin if empty
- processInfo
- identificationKey—Tasks can be queried based on the identification key from the TaskQueryService

Creating a Task

The task object model is available in the package
oracle.bpel.services.workflow.task.model

To create objects in this model, use the ObjectFactory class.

Creating a Payload Element in a Task

The task payload can contain multiple payload message attributes. Since the payload is not well defined until the task definition, the Java object model for the task does not contain strong type objects for the client payload. The task payload is represented by the AnyType Java object. The AnyType Java object is created with an XML element whose root is payload in the namespace
http://xmlns.oracle.com/bpel/workflow/task

The payload XML element contains all the other XML elements in it. Each XML element defines a message attribute.

Example 16–4 shows how to set a task payload.

**Example 16–4 Setting a Task Payload**

```java
import oracle.bpel.services.workflow.task.model.AnyType;
import oracle.bpel.services.workflow.task.model.ObjectFactory;
import oracle.bpel.services.workflow.task.model.Task;
..........
Document document = //createXMLDocument
Element payloadElem = document.createElementNS("http://xmlns.oracle.com/bpel/workflow/task", "payload");
Element orderElem = document.createElementNS("http://xmlns.oracle.com/pcbpel/test/order", "order");
Element child = document.createElementNS("http://xmlns.oracle.com/pcbpel/test/order", "id");
child.appendChild(document.createTextNode("1234567"));
orderElem.appendChild(child);
payloadElem.appendChild(orderElem);
document.appendChild(payloadElem);

task.setPayloadAsElement(payloadElem);
```

16-60 Oracle BPEL Process Manager Developer’s Guide
Example 16–5 Initiating a Vacation Request Task Programmatically

```java
// create task object
ObjectFactory objectFactory = new ObjectFactory();
Task task = objectFactory.createTask();
// set title
task.setTitle("Vacation request for jcooper");
// set creator
task.setCreator("jcooper");
// set task definition URI
String taskDefinitionURI = "http://localhost:9700/orabpel/default/VacationRequest/1.0/
VacationApproval/VacationApproval.task";
// create and set payload
Document document = XMLUtil.createDocument();
Element payloadElem = document.createElementNS(TASK_NS, "payload");
Element vacationRequestElem = document.createElementNS(VACATION_REQUEST_NS, "VacationRequestProcessRequest");
Element creatorChild = document.createElementNS(VACATION_REQUEST_NS, "creator");
creatorChild.appendChild(document.createTextNode("jcooper"));
vacationRequestElem.appendChild(creatorChild);
Element fromDateChild = document.createElementNS(VACATION_REQUEST_NS, "fromDate");
fromDateChild.appendChild(document.createTextNode("2006-08-05T12:00:00"));
vacationRequestElem.appendChild(fromDateChild);
Element toDateChild = document.createElementNS(VACATION_REQUEST_NS, "toDate");
toDateChild.appendChild(document.createTextNode("2006-08-08T12:00:00"));
vacationRequestElem.appendChild(toDateChild);
Element reasonChild = document.createElementNS(VACATION_REQUEST_NS, "reason");
reasonChild.appendChild(document.createTextNode("Hunting"));
vacationRequestElem.appendChild(reasonChild);
payloadElem.appendChild(vacationRequestElem);
document.appendChild(payloadElem);

IWorkflowServiceClient workflowServiceClient = WorkflowServiceClientFactory.getWorkflowServiceClient();
ITaskService taskService = workflowServiceClient.getTaskService();
IInitiateTaskResponse iInitiateTaskResponse = taskService.initiateTask(task);
Task retTask = iInitiateTaskResponse.getTask();
System.out.println("Initiated: " + retTask.getSystemAttributes().getTaskNumber() + " - " + retTask.getSystemAttributes().getTaskId());
```

Note: The AnyType.getContent() element returns an unmodifiable list of XML elements. You cannot add other message attributes to the list.
return retTask;

See “Vacation Request Example” on page 15-87 for more information.

Writing a Worklist Application Using the HelpDeskUI Sample

The following example shows how to modify the help desk interface that is part of the HelpDeskServiceRequest demo found at

SOA_Oracle_home/bpel/samples/demos/HelpDeskServiceRequest/HelpDeskUI

To write a Worklist Application

1. Create the workflow context by authenticating the user.

   // get workflow service client
   IWorkflowServiceClient wfSvcClient =
      WorkflowServiceClientFactory.getWorkflowServiceClient
      (WorkflowServiceClientFactory.JAVA_CLIENT);

   // get the workflow context
   IWorkflowContext wfcCtx =
      wfSvcClient.getTaskQueryService().authenticate(userId, pwd,
         oracle.tip.pc.services.identity.config.ISConfiguration.getDefaultRealmName(),
         null);

   This is Step 3 in “Building Clients for Workflow Services” on page 16-54.

   The login.jsp file of HelpDeskServiceRequest uses the preceding API to authenticate the user and create a workflow context. After the user is authenticated, the statusPage.jsp file displays the tasks assigned to the logged-in user. Example 16-6 shows sample code from the login.jsp file.

Example 16-6 Login.jsp

```html
<%@ page import="javax.servlet.http.HttpSession"
  import="oracle.bpel.services.workflow.client.IWorkflowServiceClient"
  import="oracle.bpel.services.workflow.client.WorkflowServiceClientFactory"
  import="java.util.Set"
  import="java.util.Iterator"
  import="oracle.bpel.services.workflow.verification.IWorkflowContext"
  import="oracle.tip.pc.services.identity.config.ISConfiguration"%>

<%@ page contentType="text/html;charset=windows-1252"%>

<html>
<head>
<title>Help desk request login page</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1"></head>

<body bgcolor="#F0F0F0" text="#000000" style="font: 12px verdana; line-height:18px">
<center>
<div style="width:640px;padding:15px;border-width: 10px; border-color: #87b4d9; border-style: solid; background-color:white; text-align:left">

   <!-- Page Header, Application banner, logo + user status -->
   <jsp:include page="banner.jsp"/>

   <!-- Initialize Meta Information -->
```
Welcome to the HelpDesk application

String redirectPrefix = "/HelpDeskUI/";
// Ask the browser not to cache the page
response.setHeader("Pragma", "no-cache");
response.setHeader("Cache-Control", "no-cache");
HttpSession httpSession = request.getSession(false);
if (httpSession != null) {
    IWorkflowContext ctx = (IWorkflowContext) httpSession.getAttribute("workflowContext");
    if (ctx != null) {
        response.sendRedirect(redirectPrefix + "statusPage.jsp");
    } else {
        String authFailedStr = request.getParameter("authFailed");
        boolean authFailed = false;
        if ("true".equals(authFailedStr)) {
            authFailed = true;
        } else {
            authFailed = false;
        }
    }
    if (!authFailed) {
        //Get page parameters:
        String userId="";
        if(request.getParameter("userId") != null) {
            userId = request.getParameter("userId");
        }
        String pwd="";
        if(request.getParameter("pwd") != null) {
            pwd = request.getParameter("pwd");
        }
        if(userId != null && (!"".equals(userId.trim()))) {
            if (pwd != null && (!"".equals(pwd.trim()))) {
                try {
                    HttpSession userSession = request.getSession(true);
                    IWorkflowServiceClient wfSvcClient = WorkflowServiceClientFactory.getWorkflowServiceClient(WorkflowServiceClientFactory.JAVA_CLIENT);
                    IWorkflowContext wfCtx = wfSvcClient.getTaskQueryService().authenticate(userId, pwd, oracle.tip.pc.services.identity.config.ISConfiguration.getDefaultRealmName(), null);
                    httpSession.setAttribute("workflowContext", wfCtx);
                    response.sendRedirect(redirectPrefix + "statusPage.jsp");
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        }
    } else {
        // Redirect to login page
        response.sendRedirect(redirectPrefix + "login.jsp");
    }
}
catch (Exception e) {
    String worklistServiceError = e.getMessage();
    response.sendRedirect(redirectPrefix + "login.jsp?authFailed=true");
    out.println("error is " + worklistServiceError);
} else {
    out.println("Authentication failed");
}
}

<form action='<%= request.getRequestURI() %>' method="post">
<table cellspacing="2" cellpadding="3" border="0" width="30%" align="center">
<tr>
    <td>Username</td>
    <td><input type="text" name="userId"/></td>
</tr>
<tr>
    <td>Password</td>
    <td><input type="password" name="pwd"/></td>
</tr>
<tr>
    <td><input type="submit" value="Submit"/></td>
</tr>
</table>
</form>
</div>
</center>
</body>
</html>

2. Query tasks using the queryTask API from TaskQueryService.

   //add list of attributes to be queried from the task
List displayColumns = new ArrayList();
displayColumns.add("TASKNUMBER");
displayColumns.add("TITLE");
displayColumns.add("PRIORITY");
displayColumns.add("STATE");
displayColumns.add("UPDATEDDATE");
displayColumns.add("UPDATEDBY");
displayColumns.add("CREATOR");
displayColumns.add("OUTCOME");
displayColumns.add("CREATEDDATE");
displayColumns.add("ASSIGNEEUSERS");
displayColumns.add("ASSIGNEEGROUPS");
// get the list of tasks
Building Clients for Workflow Services

List tasks = wfsvcClient.getTaskQueryService().queryTasks(wfCtx,
    displayColumns,
    null,
    ITaskQueryService.ASSIGNMENT_FILTER_MY_AND_GROUP,
    null,
    null,
    0,
    0);
// create listing page by using above tasks
//add href links to title to display details of the task by passing taskId
// as input parameter
Use getTaskDetailsById(IWorkflowContext wftx, String taskId);

This is Step 4 in "Building Clients for Workflow Services" on page 16-54.
The statusPage.jsp file of HelpDeskServiceRequest is used to display the
status of help desk requests. Example 16–7 shows the statusPage.jsp example
code.

Example 16–7 statusPage.jsp

```html
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<%@ page import="oracle.tip.pc.services.identity.BPMAuthorizationService,
     oracle.bpel.services.workflow.verification.IWorkflowContext,
     oracle.tip.pc.services.common.ServiceFactory,
     oracle.bpel.services.workflow.client.IWorkflowServiceClient,
     oracle.bpel.services.workflow.client.WorkflowServiceClientFactory,
     oracle.bpel.services.workflow.query.ITaskQueryService,
     oracle.bpel.services.workflow.task.model.Task,
     oracle.bpel.services.workflow.task.model.IdentityType,
     oracle.tip.pc.services.identity.BPMUser,
     java.util.List,
     java.util.Calendar,
     java.text.SimpleDateFormat,
     java.util.ArrayList"%>
<%@ page contentType="text/html;charset=UTF-8"%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/>
<title>RequestPage</title>
<style TYPE="text/css">
Body, Form, Table, Textarea, Select, Input, Option
{
    font-family : tahoma, verdana, arial, helvetica, sans-serif;
    font-size : 9pt;
}
table.banner
{
    background-color: #eaeff5;
}
tr.userInfo
{
    background-color: #eaeff5;
}
tr.problemInfo
{
    background-color: #87b4d9;
}
</style>
</head>
<body>RequestPage</body>
</html>
```

Worklist Application 16-65
Building Clients for Workflow Services

```java
BPMUser bpmUser = null;
String redirectPrefix = request.getContextPath() + "/\nIWorkflowContext ctx = null;
if (httpSession != null) {
    ctx = (IWorkflowContext) httpSession.getAttribute("workflowContext");
    if (ctx != null) {
        bpmUser = getAuthorizationService(ctx.getIdentityContext()).lookupUser(ctx.getUser());
    } else {
        response.sendRedirect(redirectPrefix + "login.jsp");
        return;
    }
} else {
    response.sendRedirect(redirectPrefix + "login.jsp");
    return;
}
if (bpmUser == null) {
    response.sendRedirect(redirectPrefix + "login.jsp");
    return;
}
String status = (String) httpSession.getAttribute("requeststatus");
if (status != null && !status.equals("")) {
    response.sendRedirect(redirectPrefix + "login.jsp");
    return;
}
httpSession.setAttribute("requeststatus", null);
IWorkflowServiceClient wfSvcClient = WorkflowServiceClientFactory.getWorkflowServiceClient(
    WorkflowServiceClientFactory.JAVA_CLIENT);
List displayColumns = new ArrayList();
displayColumns.add("TASKNUMBER");
displayColumns.add("TITLE");
displayColumns.add("PRIORITY");
displayColumns.add("STATE");
displayColumns.add("UPDATEDDATE");
displayColumns.add("UPDATEDBY");
displayColumns.add("CREATOR");
displayColumns.add("OUTCOME");
```
displayColumns.add("CREATEDDATE");
displayColumns.add("ASSIGNEEUSERS");
displayColumns.add("ASSIGNEEGROUPS");
List tasks = wfSvcClient.getTaskQueryService().queryTasks
 (ctx,
displayColumns,
null,
ITaskQueryService.ASSIGNMENT_FILTER_CREATOR,
null,
null,
null,
0,
0);

<div style="text-align:left;color:green">

Previous help desk request
</div>
</div>
</div>
</div>
</div>
</div>
</div>
</div>

```jsp
SimpleDateFormat dflong = new SimpleDateFormat("MM/dd/yy hh:mm a");
for(int i = 0 ; i < tasks.size() ; i ++)
{
    Task task = (Task)tasks.get(i);
    int taskNumber = task.getSystemAttributes().getTaskNumber();
    String title = task.getTitle();
    int priority = task.getPriority();
    String assignee = getAssigneeString(task);
    Calendar createdDate = task.getSystemAttributes().getCreatedDate();
    Calendar updateDate = task.getSystemAttributes().getUpdatedDate();
    String updatedBy = task.getSystemAttributes().getUpdatedBy().getId();
    String state = task.getSystemAttributes().getState();
    String outcome = task.getSystemAttributes().getOutcome();
    if(outcome == null) outcome = "";
    String titleLink = "http://" + request.getServerName() +
                        "" + request.getServerPort() +
                        "/integration/worklistapp/TaskDetails?taskId=" +
                        task.getSystemAttributes().getTaskId();

```
Building Clients for Workflow Services

<td><%=dflong.format(updateDate.getTime())%></td>
<td><%=updatedBy%></td>
<td><%=state%></td>
<td><%=outcome%></td>
<tr>
<%
}
%>
</table>
</div>
<%!
private BPMAuthorizationService getAuthorizationService(String identityContext)
{
BPMAuthorizationService authorizationService =
ServiceFactory.getAuthorizationServiceInstance();
if (identityContext != null)
authorizationService = ServiceFactory.getAuthorizationServiceInstance(identityContext);
return authorizationService;
}
private String getAssigneeString(Task task) throws Exception
{
List assignees = task.getSystemAttributes().getAssigneeUsers();
StringBuffer buffer = null;
for(int i = 0 ; i < assignees.size() ; i++)
{
IdentityType type = (IdentityType)assignees.get(i);
String name = type.getId();
if(buffer == null)
{
buffer = new StringBuffer();
}
else
{
buffer.append(",");
}
buffer.append(name).append("(U)");
}
assignees = task.getSystemAttributes().getAssigneeGroups();
for(int i = 0 ; i < assignees.size() ; i++)
{
IdentityType type = (IdentityType)assignees.get(i);
String name = type.getId();
if(buffer == null)
{
buffer = new StringBuffer();
}
else
{
buffer.append(",");
}
buffer.append(name).append("(G)");
}
if(buffer == null)
{
return "";
}
else
{
return buffer.toString();

16-68 Oracle BPEL Process Manager Developer’s Guide


Summary

This chapter describes how to access a user’s tasks, view task details, and perform actions on the tasks in the sample Oracle BPEL Worklist Application. It also discusses how you can create and share custom views, manage user and group rules, customize task display settings, and perform administrative tasks such as flex field mapping and application customization. Instructions are provided for customizing the Worklist Application (including a number of language settings) and for building your own Worklist Application using the workflow service APIs.
Using sensors, you can specify BPEL activities, variables, and faults that you want to monitor during run time. This chapter describes how to use and set up sensors for a BPEL process. This chapter also describes how to create sensor actions in Oracle BPEL Process Manager to publish sensor data as data objects in an Oracle BAM Server.

This chapter contains the following topics:
- Use Cases for Sensors
- Overview of Sensor Concepts
- Implementing Sensors and Sensor Actions in Oracle JDeveloper
- Sensors and Oracle BPEL Control
- Sensor Integration with Oracle Business Activity Monitoring
- Sensor Public Views
- Sensor Actions XSD File
- Summary

Use Cases for Sensors

Using sensors is demonstrated in the sample 125.ReportsSchema. The sample uses sensors to identify key data during an employee update process and a sensor action to publish information about the update to the database:

See: SOA_Oracle_Home\bpel\samples\tutorials\125.ReportsSchema

Inserting sensors on activities is also demonstrated in the OrderBooking tutorial.

See: Oracle BPEL Process Manager Order Booking Tutorial

Overview of Sensor Concepts

You can define the following types of sensors, either through Oracle JDeveloper or manually by providing sensor configuration files.

- Activity sensors
  Activity sensors are used to monitor the execution of activities within a BPEL process. For example, they can be used to monitor the execution time of an invoke activity or how long it takes to complete a scope. Along with the activity sensor, you can also monitor variables of the activity.

Sensors 17-1
Variable sensors

Variable sensors are used to monitor variables (or parts of a variable) of a BPEL process. For example, variable sensors can be used to monitor the input and output data of a BPEL process.

Fault sensors

Fault sensors are used to monitor BPEL faults.

You typically add or edit sensors as part of the BPEL modeling of activities, faults, and variables.

When you model sensors in Oracle JDeveloper, two new files are created as part of the BPEL process suitcase:

- **sensor.xml**—contains the sensor definitions of a BPEL process
- **sensorAction.xml**—contains the sensor action definitions of a BPEL process

See "Configuring Sensors" on page 17-3 and "Configuring Sensor Actions" on page 17-6 for how these files are created.

After you define sensors for a BPEL process, you must configure sensor actions to publish the data of the sensors to an endpoint. You can publish sensor data to the BPEL reports schema, which is located in the BPEL dehydration store, to a JMS queue or topic, or to a custom Java class.

The following information is required for a sensor action:

- **Name**
- **Publish type**

  The publish type specifies the destination where the sensor data must be presented. You can configure the following publish types:
  - Database—used to publish the sensor data to the reports schema in the database. The sensor data can then be queried using SQL.
  - JMSQueue—used to publish the sensor data to a JMS queue
  - JMSTopic—used to publish the sensor data to a JMS topic
  - Custom—used to publish the data to a custom Java class
  - JMS Adapter—uses the JMS adapter to publish to remote queues or topics and a variety of different JMS providers. The JMS Queue and JMS Topic publish types only publish to local JMS destinations.

- **List of sensors**—the sensors for a sensor action

**Implementing Sensors and Sensor Actions in Oracle JDeveloper**

In Oracle JDeveloper, sensor actions and sensors are displayed as part of the process tree structure, as shown in Figure 17-1.
You typically add or edit sensors as part of the BPEL modeling of activities, faults, and variables. You can add sensor actions by right-clicking the Sensor Actions folders and selecting Create > Sensor Action. To add activity sensors, variable sensors, or fault sensors, expand the Sensors folder, right-click the appropriate Activity, Variable, or Fault subfolder, and click Create.

Using LoanDemoPlus as an example, the following sections describe how to configure sensors and sensor actions.

Configuring Sensors

If you are monitoring the LoanFlow application, you may want to know when the getCreditRating scope is initiated, when it is completed, and, at completion, what the credit rating for the customer is. The solution is to create an activity sensor for the getCreditRating scope in Oracle JDeveloper, as shown in Figure 17-2. Activities that have sensors associated with them are identified with a magnifying glass in Oracle JDeveloper.
The Evaluation Time attribute shown in Figure 17–2 controls the point at which the sensor fires. You can select from the following:

- **Activation**—The sensor fires just before the activity is executed.
- **Completion**—The sensor fires just after the activity is executed.
- **Fault**—The sensor fires if a fault occurs during the execution of the activity. Select this value only for sensors that monitor simple activities.
- **Compensation**—The sensor fires when the associated scope activity is compensated. Select this value only for sensors that monitor scopes.
- **Retry**—The sensor fires when the associated invoke activity is retried.
- **All**—Monitoring occurs during all of the preceding phases.

A new entry is created in the `sensor.xml` file, as follows:

```xml
<(sensor sensorName="CreditRatingSensor"
  class="oracle.tip.pc.services.reports.dca.agents.BpelActivitySensorAgent"
  kind="activity"
  target="getCreditRating">
  <activityConfig evalTime="all">
    <variable outputNamespace="http://www.w3.org/2001/XMLSchema"
      outputDataType="int"
      target="$crOutput/payload//services:rating"/>
  </activityConfig>
</sensor>
```

If you want to record all the incoming loan requests, create a variable sensor for the variable `input`, as shown in Figure 17–3.
Implementing Sensors and Sensor Actions in Oracle JDeveloper

Figure 17–3  Creating a Variable Sensor

A new entry is created in the sensor.xml file, as follows:

```xml
<sensor sensorName="LoanApplicationSensor"
classname="oracle.tip.pc.services.reports.dca.agents.BpelVariableSensorAgent"
kind="variable"
target="$input/payload">
  <variableConfig outputNamespace="http://www.autoloan.com/ns/autoloan"
    outputDataType="loanApplication"/>
</sensor>
```

If you want to monitor faults from the identity service, create a fault sensor, as shown in Figure 17–4.

Figure 17–4  Creating a Fault Sensor

A new entry is created in the sensor.xml file, as follows:

```xml
<sensor sensorName="IdentityServiceFault"
classname="oracle.tip.pc.services.reports.dca.agents.BpelFaultSensorAgent"
kind="fault">
  <variableConfig outputNamespace="http://www.autoloan.com/ns/autoloan"
    outputDataType="loanApplication"/>
</sensor>
```
Configuring Sensor Actions

When you create sensors, you identify the activities, variables, and faults you want to monitor during runtime. If you want to publish the values of the sensors to an endpoint (for example, you want to publish the data of LoanApplicationSensor to a JMS queue), then create a sensor action, as shown in Figure 17–5, and associate it with the LoanApplicationSensor.

Figure 17–5 Creating a Sensor Action

A new entry is created in the sensorAction.xml file, as follows:

```xml
<action name="BAMFeed" enabled="true" publishType="JMSQueue" publishTarget="%jms/bamTopic">
  <sensorName>LoanApplicationSensor</sensorName>
  <property name="JMSConnectionFactory">jms/QueueConnectionFactory</property>
</action>
```

If you want to publish the values of LoanApplicationSensor and CreditRatingSensor to the reports schema in the database, create an additional sensor action, as shown in Figure 17–6, and associate it with both CreditRatingSensor and LoanApplicationSensor.

Figure 17–6 Creating an Additional Sensor Action

A new entry is created in the sensorAction.xml file, as follows:

```xml
<action name="BAMFeed" enabled="true" publishType="JMSQueue" publishTarget="%jms/bamTopic">
  <sensorName>LoanApplicationSensor</sensorName>
  <sensorName>CreditRatingSensor</sensorName>
  <property name="JMSConnectionFactory">jms/QueueConnectionFactory</property>
</action>
```
Implementing Sensors and Sensor Actions in Oracle JDeveloper

```xml
<action name="PersistingAction"
  enabled="true"
  publishType="BPELReportsSchema">
  <sensorName>LoanApplicationSensor</sensorName>
  <sensorName>CreditRatingSensor</sensorName>
</action>
```

The data of one sensor can be published to multiple endpoints. In the two preceding code samples, the data of LoanApplicationSensor is published to a JMS queue and to the reports schema in the database.

If you want to monitor loan requests for which the loan amount is greater than $100,000, you can create a sensor action with a filter, as shown in Figure 17–7.

![Figure 17–7 Creating a Sensor Action with a Filter](image)

A new entry is created in the sensorAction.xml file, as follows:

```xml
<action name="BigMoneyBAMAction"
  enabled='true'
  filter="boolean(/s:actionData/s:payload
    /s:variableData/a:autoloan
      /a:loanAmount > 100000)"
  publishType="JMSQueue"
  publishTarget="jms/bigMoneyQueue">
  <sensorName>LoanApplicationSensor</sensorName>
  <property name="JMSConnectionFactory">
    jms/QueueConnectionFactory
  </property>
</action>
```

Note:
- You must specify all the namespaces that are required to configure an action filter in the sensor action configuration file.
- You must specify the filter as a Boolean XPath expression.

If you have special requirements for a sensor action that cannot be accomplished by using the built-in publish types (database, JMS queue, JMS topic, and JMS Adapter), then you can create a sensor action with the custom publish type, as shown in Figure 17–8. The name in the Publish Target field denotes a fully qualified Java class name that must be implemented.
Implementing Sensors and Sensor Actions in Oracle JDeveloper

Figure 17–8 Using the Custom Publish Type

Publishing to Remote Topics and Queues

The JMS Queue and JMS Topic publish types only publish to local JMS destinations. If you want to publish sensor data to remote topics and queues, use the JMS adapter publish type, as shown in Figure 17–9.

Figure 17–9 Using the JMS Adapter Publish Type

In addition to enabling you to publish sensor data to remote topics and queues, the JMS adapter supports a variety of different JMS providers, including:

- Third-party JMS providers such as Tibco JMS, IBM WebSphere MQ JMS, and SonicMQ
- Oracle Enterprise Messaging Service (OEMS) providers such as memory/file and database

If you select the JMS Adapter publish type, you must create an entry in the oc4j-ra.xml file. Use the JMS connection name property in the Sensor Actions dialog to point to the proper entry in the oc4j-ra.xml file.

See Also: Oracle Application Server Adapter for Files, FTP, Databases, and Enterprise Messaging User’s Guide for details about the JMS adapter

Creating a Custom Data Publisher

To create a custom data publisher, double-click your BPEL project in Oracle JDeveloper and do the following:

2. Browse and select SOA_Oracle_Home\bpel\lib\orabpel.jar.
3. Create a new Java class.  
The package and class name must match the publish target name of the sensor action.

4. Implement the `com.oracle.bpel.sensor.DataPublisher` interface.  
   This updates the source file and fills in the methods and import statements of the `DataPublisher` interface.

5. Using the Oracle JDeveloper editor, implement the publish method of the `DataPublisher` interface, as shown in the following sample custom data publisher class.
6. Ensure that the class compiles successfully.

The next time that you deploy the BPEL process, the Java class is added to the
BPEL suitcase and deployed to Oracle BPEL Process Manager.
Registering the Sensors and Sensor Actions in bpel.xml

Oracle JDeveloper automatically updates the process deployment file bpel.xml to include appropriate properties for sensors and sensor actions, as follows:

```xml
<configurations>
  <property name="sensorLocation">sensor.xml</property>
  <property name="sensorActionLocation">sensorAction.xml</property>
  <property name="SLACompletionTime">P0YT1.5S</property>
</configurations>
```

You can specify additional properties with `<property name= ...>`, as shown in the preceding code sample.

Sensors and Oracle BPEL Control

The console provides support to view the metadata of sensors and sensor actions as well as the sensor data created as part of the process execution. Access Oracle BPEL Control at http://localhost:port/BPELConsole

You can also select Start > All Programs > Oracle - Oracle_Home > Oracle BPEL Process Manager > BPEL Control.

Viewing Sensor and Sensor Action Definitions

After the BPEL process is deployed to Oracle BPEL Process Manager, you can view the definitions of sensors and sensor actions without going back to Oracle JDeveloper. In Oracle BPEL Control, click the BPEL Processes tab and choose the process for which you want to see sensor definitions. Click the Sensors link. A page similar to Figure 17–10 is displayed.

Note: Ensure that additional Java libraries needed to implement the data publisher are in the CLASSPATH of the Oracle BPEL Server.

Oracle BPEL Process Manager can execute multiple process instances simultaneously, so ensure that the code in your data publisher is thread safe, or add appropriate synchronization blocks. To guarantee high throughput, do not use shared data objects that require synchronization.
Figure 17–10  Sensor Data on the BPEL Processes Tab of Oracle BPEL Control

<table>
<thead>
<tr>
<th>Sensor Definition</th>
<th>Name</th>
<th>Description</th>
<th>Action</th>
<th>Variables</th>
<th>Query</th>
<th>Namespace</th>
<th>Data Type</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Viewing Sensor Data

The console provides support to monitor sensors for which a `BpelReportsSchema` sensor action is defined. In Oracle BPEL Control, click the Instances tab and choose the process instance for which you want to see the sensor data created as the result of process execution. A page similar to Figure 17–11 is displayed.
Oracle Business Activity Monitoring (BAM) enables you to monitor business services and processes in an enterprise, correlate key performance indicators (KPIs), and change business processes or take corrective actions if the business environment changes.

Oracle BAM enables you to build real-time operational dashboards and monitoring and alerting applications over the Web. Using this technology, you build interactive, real-time dashboards and proactive alerts to monitor business services and processes.

You can create sensor actions in Oracle BPEL Process Manager to publish sensor data as data objects on an Oracle BAM Server. When you create the sensor action, you can select an Oracle BPEL Process Manager variable or activity sensor that you want to monitor and the data object in Oracle BAM Server in which you want to publish the data.
Sensor Integration with Oracle Business Activity Monitoring

This section contains the following topics:

- Creating a Connection to Oracle BAM Server
- Creating a Sensor
- Creating a BAM Sensor Action

These instructions assume you have installed and configured Oracle BAM.

See Also: Oracle Business Activity Monitoring Administrator’s Guide

Creating a Connection to Oracle BAM Server

You must create a connection to Oracle BAM Server to browse the available data objects. This connection information is automatically used during deployment.

1. Select Connection Navigator from the View main menu in Oracle JDeveloper.
2. Right click BAM Server.
4. Click Next on the Welcome page.
5. Provide a name for connecting to the server.
6. Click Next.
7. Enter the following connection information about the Oracle BAM Server host.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host Name</td>
<td>Enter the name of the host on which Oracle BAM Server is installed. Depending on your organization’s security policy, the fully-qualified host name may be required.</td>
</tr>
<tr>
<td>Port Number</td>
<td>Enter the port number or accept the default value of 80.</td>
</tr>
<tr>
<td>User Name</td>
<td>Enter your Windows domain user name. Oracle BAM Server uses the Windows domain for authentication.</td>
</tr>
<tr>
<td>Password</td>
<td>Enter the password of the domain user name.</td>
</tr>
<tr>
<td>Domain Name</td>
<td>Enter the domain name in which the Oracle BAM Server host is located. This field is case sensitive. If you do not enter the correct case sensitive name, you receive an authentication failure error.</td>
</tr>
<tr>
<td>Use secure HTTP protocol</td>
<td>Select this check box if you want to use secure HTTP (HTTP/S) to connect to the Oracle BAM Server. Otherwise, HTTP is used.</td>
</tr>
</tbody>
</table>

8. Click Next.
9. Test the connection by clicking Test Connection. If the connection was successful, the following message appears:

Success.
10. Click Finish.

Oracle JDeveloper reserves the following property names. These property names define values for the Oracle BAM Server connection you just created.

- `bamservlet.hostname`
- `bamservlet.protocol`
- `bamservlet.username`
- `bamservlet.password`
- `bamservlet.port`
- `bamservlet.domain`

These property names are added in the Preferences tab of the Deployment Descriptor Properties window. If your BPEL process uses a BAM sensor action and you want run time to use a different Oracle BAM Server than the one used during design time, you must update these values prior to BPEL process deployment. If you have already deployed the process, then you can use Oracle BPEL Control to update these values.

**See Also:**
- Chapter 19, "BPEL Process Deployment and Domain Management"
- Appendix C, "Deployment Descriptor Properties"

### Creating a Sensor

You must create one of the following types of sensors prior to creating a BAM sensor action:

- A variable sensor. Since you map the sensor data to Oracle BAM Server data objects, only one variable is allowed for the sensor. If the variable has message parts, then there should be only one message part. This variable must not be defined inline in the WSDL. Only XSD element definitions are supported.

- An activity sensor containing exactly one sensor variable.

**See Also:** "Implementing Sensors and Sensor Actions in Oracle JDeveloper" on page 17-2 for instructions on creating sensors

### Creating a BAM Sensor Action

When you create the sensor action, you select the BPEL variable or activity sensor that you want to monitor and the data object in Oracle BAM Server in which you want to publish the sensor data.

1. Right click **Sensor Actions** in the **Structure** section of Oracle JDeveloper.
2. Select **Create > BAM Sensor Action**.

The Create Sensor Action window appears. You create BAM sensor actions to publish sensor data to data objects on Oracle BAM Server.

Sensors 17-15
3. Enter the following details:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action Name</td>
<td>Enter a unique and recognizable name for the sensor action.</td>
</tr>
<tr>
<td>Select Sensor</td>
<td>Select a BPEL sensor to monitor. This is the sensor that you created in &quot;Creating a Sensor&quot; on page 17-15 for mapping sensor data to a data object in Oracle BAM Server.</td>
</tr>
<tr>
<td>Data Object</td>
<td>Click the flashlight icon to open the BAM Data Object Chooser window to select the data object in Oracle BAM Server in which you want to publish the sensor data. You must have already created a connection to Oracle BAM Server in order to select data objects.</td>
</tr>
<tr>
<td>Enable Batching</td>
<td>The data cached by default by the Oracle BAM component of the Oracle BPEL Process Manager run time is flushed (sent) to Oracle BAM Server periodically. The decision to periodically send the data is based on upper and lower limit parameter settings on the Set Batch Parameters window. The Oracle BAM component may decide to send data prior to a batch timeout if the cache has a number of data objects between the lower and upper limit values. Disable batching by unselecting this check box. To modify the batch parameters, click Set Batch Parameters. See Step 4 on page 17-17 for additional details.</td>
</tr>
</tbody>
</table>
If you want to specify custom batch parameter settings, click Set Batch Parameters. The Set Batch Parameters window appears.

5. Deselect the Use Default Batch Parameters check box and provide customized values in the fields below. If you provide customized values and then select this check box again, the settings revert to the default values.

### Field Description

#### Batch size lower limit
Use the default value of 1000 or specify a lower batch limit. This parameter controls the minimum number of rows in the cache. With this parameter, the data remains in the Oracle BPEL Process Manager run-time cache until the queue size reaches at least this limit or a timeout occurs.
Click OK to close the Set Batch Parameters window and the Create Sensor Action window.

Sensor Public Views

The sensor framework of Oracle BPEL Process Manager provides the functionality to persist sensor values created by processing BPEL instances in a relational schema stored in the dehydration store of Oracle BPEL Process Manager. The data is used to display the sensor values of a process instance in Oracle BPEL Control.

A set of public views is exposed to allow SQL access to sensor values from literally any application interested in the data.

BPM Schema

The database publisher persists the sensor data in a predefined relational schema in the database. The following public views can be used from a client (Oracle Warehouse, OracleAS Portal, and so on) to query the sensor values using SQL.

Note: In Table 17–1 through Table 17–5, the Indexed or Unique? column provides unique index names and the order of the attributes. For example, U1,2 means that the attribute is the second one in a unique index named U1. PK means primary key.

Table 17–1 BPM_SCHEMA_INSTANCES
This view provides an overview of all the process instances of Oracle BPEL Process Manager.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INSTANCE_KEY</td>
<td>Unique instance ID</td>
</tr>
<tr>
<td>NAME</td>
<td>Name of the BPEL process</td>
</tr>
<tr>
<td>REVISION</td>
<td>Revision of the BPEL process</td>
</tr>
<tr>
<td>DOMAIN_ID</td>
<td>Oracle BPEL Process Manager domain name</td>
</tr>
<tr>
<td>TITLE</td>
<td>User-defined title of the BPEL process</td>
</tr>
<tr>
<td>STATE</td>
<td>State of the BPEL process</td>
</tr>
<tr>
<td>STATUS TEXT</td>
<td>Text presentation of the state attribute</td>
</tr>
</tbody>
</table>

Field | Description
---|---
Batch size upper limit | Use the default value of 5000 or specify an upper batch limit. This parameter controls the maximum number of rows in the cache. With this parameter, the Oracle BPEL Process Manager runtime flushes the data to Oracle BAM Server prior to the upper limit being reached.
Batch timeout (milliseconds) | Specify the timeout in minutes. The default value is 50 milliseconds. When the timeout occurs, the BPEL runtime flushes any data in the queue to Oracle BAM Server.
Table 17–1 (Cont.) BPEL_PROCESS_INSTANCES View

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>SQL Type</th>
<th>Attribute Size</th>
<th>Indexed or Unique?</th>
<th>Null?</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>NUMBER</td>
<td>--</td>
<td>PK</td>
<td>N</td>
<td>Unique ID</td>
</tr>
<tr>
<td>INSTANCE_KEY</td>
<td>NUMBER</td>
<td>--</td>
<td>U1,1</td>
<td>N</td>
<td>ID of process instance</td>
</tr>
<tr>
<td>BPEL_PROCESS_NAME</td>
<td>VARCHAR2</td>
<td>100</td>
<td>--</td>
<td>N</td>
<td>Name of the BPEL process</td>
</tr>
<tr>
<td>BPEL_PROCESS_REVISION</td>
<td>VARCHAR2</td>
<td>50</td>
<td>--</td>
<td>N</td>
<td>Revision of the BPEL process</td>
</tr>
<tr>
<td>DOMAIN_ID</td>
<td>VARCHAR2</td>
<td>50</td>
<td>--</td>
<td>N</td>
<td>Oracle BPEL Process Manager domain name</td>
</tr>
<tr>
<td>SENSOR_NAME</td>
<td>VARCHAR2</td>
<td>100</td>
<td>U1,2</td>
<td>N</td>
<td>The name of the sensor that fired</td>
</tr>
<tr>
<td>SENSOR_TARGET</td>
<td>VARCHAR2</td>
<td>256</td>
<td>--</td>
<td>N</td>
<td>The target of the fired sensor</td>
</tr>
</tbody>
</table>

Table 17–2 BPEL_ACTIVITY_SENSOR_VALUES View

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>SQL Type</th>
<th>Attribute Size</th>
<th>Indexed or Unique?</th>
<th>Null?</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>NUMBER</td>
<td>--</td>
<td>U1,2</td>
<td>N</td>
<td>Unique ID</td>
</tr>
<tr>
<td>INSTANCE_KEY</td>
<td>NUMBER</td>
<td>--</td>
<td>U1,1</td>
<td>N</td>
<td>ID of process instance</td>
</tr>
<tr>
<td>BPEL_PROCESS_NAME</td>
<td>VARCHAR2</td>
<td>100</td>
<td>--</td>
<td>N</td>
<td>Name of the BPEL process</td>
</tr>
<tr>
<td>BPEL_PROCESS_REVISION</td>
<td>VARCHAR2</td>
<td>50</td>
<td>--</td>
<td>N</td>
<td>Revision of the BPEL process</td>
</tr>
<tr>
<td>DOMAIN_ID</td>
<td>VARCHAR2</td>
<td>50</td>
<td>--</td>
<td>N</td>
<td>Oracle BPEL Process Manager domain name</td>
</tr>
<tr>
<td>SENSOR_NAME</td>
<td>VARCHAR2</td>
<td>100</td>
<td>U1,2</td>
<td>N</td>
<td>The name of the sensor that fired</td>
</tr>
<tr>
<td>SENSOR_TARGET</td>
<td>VARCHAR2</td>
<td>256</td>
<td>--</td>
<td>N</td>
<td>The target of the fired sensor</td>
</tr>
</tbody>
</table>

BPEL_ACTIVITY_SENSOR_VALUES
This view contains all the activity sensor values of the monitored BPEL processes.
Table 17–3 BPEL_FAULT_SENSOR_VALUES View

This view contains all the fault sensor values.

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>SQL Type</th>
<th>Attribute Size</th>
<th>Indexed or Unique?</th>
<th>Null?</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>NUMBER</td>
<td>--</td>
<td>PK</td>
<td>N</td>
<td>Unique ID</td>
</tr>
<tr>
<td>INSTANCE_KEY</td>
<td>NUMBER</td>
<td>--</td>
<td>U1,1</td>
<td>N</td>
<td>BPEL process ID</td>
</tr>
<tr>
<td>BPEL_PROCESS_NAME</td>
<td>VARCHAR2</td>
<td>100</td>
<td>--</td>
<td>N</td>
<td>Name of the BPEL process</td>
</tr>
<tr>
<td>BPEL_PROCESS_REVISION</td>
<td>VARCHAR2</td>
<td>50</td>
<td>--</td>
<td>N</td>
<td>Revision of the BPEL process</td>
</tr>
<tr>
<td>DOMAIN_ID</td>
<td>VARCHAR2</td>
<td>50</td>
<td>--</td>
<td>N</td>
<td>Oracle BPEL Process Manager domain name</td>
</tr>
<tr>
<td>SENSOR_NAME</td>
<td>VARCHAR2</td>
<td>100</td>
<td>U1,2</td>
<td>N</td>
<td>The name of the sensor that fired</td>
</tr>
<tr>
<td>SENSOR_TARGET</td>
<td>VARCHAR2</td>
<td>256</td>
<td>--</td>
<td>N</td>
<td>The target of the fired sensor</td>
</tr>
<tr>
<td>ACTION_NAME</td>
<td>VARCHAR2</td>
<td>100</td>
<td>U1,3</td>
<td>N</td>
<td>The name of the sensor action</td>
</tr>
<tr>
<td>ACTION_FILTER</td>
<td>VARCHAR2</td>
<td>256</td>
<td>--</td>
<td>Y</td>
<td>The filter of the action</td>
</tr>
<tr>
<td>CREATION_DATE</td>
<td>TIMESTAMP</td>
<td>--</td>
<td>--</td>
<td>N</td>
<td>The creation date of the activity sensor value</td>
</tr>
<tr>
<td>MODIFY_DATE</td>
<td>TIMESTAMP</td>
<td>--</td>
<td>--</td>
<td>Y</td>
<td>The time stamp of last modification</td>
</tr>
<tr>
<td>MODIFY_DATE</td>
<td>TIMESTAMP</td>
<td>--</td>
<td>--</td>
<td>Y</td>
<td>Date portion of modify_date</td>
</tr>
<tr>
<td>TS_DATE</td>
<td>DATE</td>
<td>--</td>
<td>--</td>
<td>Y</td>
<td>Hour portion of modify_date</td>
</tr>
<tr>
<td>MODIFIED_DATE</td>
<td>NUMBER</td>
<td>--</td>
<td>--</td>
<td>Y</td>
<td>Evaluation time of the activity in milliseconds</td>
</tr>
</tbody>
</table>
This view contains all the variable sensor values.

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>SQL Type</th>
<th>Attribute Size</th>
<th>Indexed or Unique?</th>
<th>Null?</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>NUMBER</td>
<td>--</td>
<td>PK</td>
<td>N</td>
<td>Unique ID</td>
</tr>
<tr>
<td>INSTANCE_KEY</td>
<td>NUMBER</td>
<td>--</td>
<td>U1,1</td>
<td>N</td>
<td>BPEL process ID</td>
</tr>
<tr>
<td>BPEL_PROCESS_NAME</td>
<td>NVARCHAR2</td>
<td>100</td>
<td>--</td>
<td>N</td>
<td>Name of the BPEL process</td>
</tr>
<tr>
<td>BPEL_PROCESS_REVISION</td>
<td>VARCHAR2</td>
<td>50</td>
<td>--</td>
<td>N</td>
<td>Revision of the BPEL process</td>
</tr>
<tr>
<td>DOMAIN_ID</td>
<td>VARCHAR2</td>
<td>50</td>
<td>--</td>
<td>N</td>
<td>Oracle BPEL Process Manager domain name</td>
</tr>
<tr>
<td>SENSOR_NAME</td>
<td>NVARCHAR2</td>
<td>100</td>
<td>U1,2</td>
<td>N</td>
<td>Name of the sensor that fired</td>
</tr>
<tr>
<td>SENSOR_TARGET</td>
<td>NVARCHAR2</td>
<td>256</td>
<td>--</td>
<td>N</td>
<td>Target of the sensor</td>
</tr>
<tr>
<td>ACTION_NAME</td>
<td>NVARCHAR2</td>
<td>100</td>
<td>U1,3</td>
<td>N</td>
<td>Name of the action</td>
</tr>
<tr>
<td>ACTION_FILTER</td>
<td>NVARCHAR2</td>
<td>256</td>
<td>--</td>
<td>Y</td>
<td>Filter of the action</td>
</tr>
<tr>
<td>ACTIVITY_NUMBER</td>
<td>NUMBER</td>
<td>--</td>
<td>--</td>
<td>Y</td>
<td>ID of corresponding activity sensor value</td>
</tr>
<tr>
<td>CREATION_DATE</td>
<td>TIMESTAMP</td>
<td>--</td>
<td>--</td>
<td>N</td>
<td>Creation date</td>
</tr>
<tr>
<td>TS_DATE</td>
<td>DATE</td>
<td>--</td>
<td>--</td>
<td>N</td>
<td>Date portion of creation_date</td>
</tr>
<tr>
<td>TS_HOUR</td>
<td>NUMBER</td>
<td>--</td>
<td>--</td>
<td>N</td>
<td>Hour portion of creation_date</td>
</tr>
<tr>
<td>VARIABLE_NAME</td>
<td>NVARCHAR2</td>
<td>256</td>
<td>--</td>
<td>N</td>
<td>The name of the BPEL variable</td>
</tr>
<tr>
<td>EVAL_POINT</td>
<td>VARCHAR2</td>
<td>30</td>
<td>--</td>
<td>Y</td>
<td>Evaluation point of the corresponding activity sensor</td>
</tr>
<tr>
<td>CRITERIA_SATISFIED</td>
<td>VARCHAR2</td>
<td>1</td>
<td>--</td>
<td>Y</td>
<td>NULL, Y, or N</td>
</tr>
<tr>
<td>TARGET</td>
<td>NVARCHAR2</td>
<td>256</td>
<td>--</td>
<td>--</td>
<td></td>
</tr>
<tr>
<td>UPDATER_NAME</td>
<td>NVARCHAR2</td>
<td>100</td>
<td>--</td>
<td>N</td>
<td>The name of the activity or event that updated the variable</td>
</tr>
<tr>
<td>UPDATER_TYPE</td>
<td>NVARCHAR2</td>
<td>100</td>
<td>--</td>
<td>N</td>
<td>The type of the BPEL activity or event</td>
</tr>
<tr>
<td>SCHEMA_NAMESPACE</td>
<td>NVARCHAR2</td>
<td>256</td>
<td>--</td>
<td>Y</td>
<td>Namespace of variable sensor value</td>
</tr>
</tbody>
</table>

**Table 17-4**  
BPEL_VARIABLE_SENSOR_VALUES View

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>SQL Type</th>
<th>Attribute Size</th>
<th>Indexed or Unique?</th>
<th>Null?</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>TS_HOUR</td>
<td>NUMBER</td>
<td>--</td>
<td>--</td>
<td>Y</td>
<td>Hour portion of modify_date</td>
</tr>
<tr>
<td>CRITERIA_SATISFIED</td>
<td>VARCHAR2</td>
<td>1</td>
<td>--</td>
<td>Y</td>
<td>NULL if no action filter specified; Y if action filter is specified and evaluates to true; N otherwise</td>
</tr>
<tr>
<td>ACTIVITY_NAME</td>
<td>NVARCHAR2</td>
<td>100</td>
<td>--</td>
<td>N</td>
<td>The name of the BPEL activity</td>
</tr>
<tr>
<td>ACTIVITY_TYPE</td>
<td>VARCHAR2</td>
<td>30</td>
<td>--</td>
<td>N</td>
<td>The type of the BPEL activity</td>
</tr>
<tr>
<td>MESSAGE</td>
<td>CLOB</td>
<td>--</td>
<td>--</td>
<td>Y</td>
<td>The fault message</td>
</tr>
<tr>
<td>TS_DATE</td>
<td>DATE</td>
<td>--</td>
<td>--</td>
<td>N</td>
<td>Date portion of modification_date</td>
</tr>
<tr>
<td>TS_HOUR</td>
<td>NUMBER</td>
<td>--</td>
<td>--</td>
<td>N</td>
<td>Hour portion of TS_DATE</td>
</tr>
<tr>
<td>VARIABLE_NAME</td>
<td>NVARCHAR2</td>
<td>256</td>
<td>--</td>
<td>N</td>
<td>The name of the BPEL variable</td>
</tr>
<tr>
<td>EVAL_POINT</td>
<td>VARCHAR2</td>
<td>30</td>
<td>--</td>
<td>Y</td>
<td>Evaluation point of the corresponding activity sensor</td>
</tr>
<tr>
<td>CRITERIA_SATISFIED</td>
<td>VARCHAR2</td>
<td>1</td>
<td>--</td>
<td>Y</td>
<td>NULL, Y, or N</td>
</tr>
<tr>
<td>TARGET</td>
<td>NVARCHAR2</td>
<td>256</td>
<td>--</td>
<td>--</td>
<td></td>
</tr>
<tr>
<td>UPDATER_NAME</td>
<td>NVARCHAR2</td>
<td>100</td>
<td>--</td>
<td>N</td>
<td>The name of the activity or event that updated the variable</td>
</tr>
<tr>
<td>UPDATER_TYPE</td>
<td>NVARCHAR2</td>
<td>100</td>
<td>--</td>
<td>N</td>
<td>The type of the BPEL activity or event</td>
</tr>
<tr>
<td>SCHEMA_NAMESPACE</td>
<td>NVARCHAR2</td>
<td>256</td>
<td>--</td>
<td>Y</td>
<td>Namespace of variable sensor value</td>
</tr>
</tbody>
</table>
Table 17–4  (Cont.) BPEL_VARIABLE SENSOR VALUES View

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>SQL Type</th>
<th>Attribute Size</th>
<th>Indexed or Unique?</th>
<th>Null?</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCHEMA_DATATYPE</td>
<td>NVARCHAR2</td>
<td>256</td>
<td>--</td>
<td>Y</td>
<td>Datatype of the variable sensor value</td>
</tr>
<tr>
<td>VALUE_TYPE</td>
<td>SMALLINT</td>
<td>--</td>
<td>--</td>
<td>N</td>
<td>The value type of the variable (corresponds to java.sql.Types values)</td>
</tr>
<tr>
<td>VARCHAR2_VALUE</td>
<td>NVARCHAR2</td>
<td>2000</td>
<td>--</td>
<td>Y</td>
<td>The value of string-like variables</td>
</tr>
</tbody>
</table>

BPEMERRORS
This view provides an overview of all errors from BPM services.

Table 17-5  BPEMERRORS View

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>SQL Type</th>
<th>Attribute Size</th>
<th>Indexed or Unique?</th>
<th>Null?</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>NUMBER</td>
<td>--</td>
<td>PK</td>
<td>N</td>
<td>Unique ID</td>
</tr>
<tr>
<td>BPEL_PROCESS_NAME</td>
<td>NVARCHAR2</td>
<td>100</td>
<td>U1,1</td>
<td>N</td>
<td>Name of the BPEL process</td>
</tr>
<tr>
<td>BPEL_PROCESS_REVISION</td>
<td>VARCHAR2</td>
<td>50</td>
<td>U1,2</td>
<td>N</td>
<td>Revision of the BPEL process</td>
</tr>
<tr>
<td>DOMAIN_ID</td>
<td>VARCHAR2</td>
<td>50</td>
<td>U1,3</td>
<td>N</td>
<td>Oracle BPEL Process Manager domain name</td>
</tr>
<tr>
<td>CREATION_DATE</td>
<td>TIMESTAMP</td>
<td>--</td>
<td>--</td>
<td>N</td>
<td>Creation date of the activity sensor value</td>
</tr>
<tr>
<td>TS_DATE</td>
<td>DATE</td>
<td>--</td>
<td>--</td>
<td>N</td>
<td>Date portion of creation_date</td>
</tr>
<tr>
<td>TS_HOUR</td>
<td>NUMBER</td>
<td>--</td>
<td>--</td>
<td>N</td>
<td>Hour portion of creation_date</td>
</tr>
<tr>
<td>ERROR_CODE</td>
<td>NUMBER</td>
<td>--</td>
<td>--</td>
<td>N</td>
<td>Error code</td>
</tr>
<tr>
<td>EXCEPTION_TYPE</td>
<td>NUMBER</td>
<td>--</td>
<td>--</td>
<td>N</td>
<td>Type of the error</td>
</tr>
<tr>
<td>EXCEPTION_SEVERITY</td>
<td>NUMBER</td>
<td>--</td>
<td>--</td>
<td>N</td>
<td>Severity of the error</td>
</tr>
<tr>
<td>EXCEPTION_NAME</td>
<td>NVARCHAR2</td>
<td>200</td>
<td>--</td>
<td>N</td>
<td>Name of the error</td>
</tr>
<tr>
<td>EXCEPTION_DESCRIPTION</td>
<td>NVARCHAR2</td>
<td>2000</td>
<td>--</td>
<td>Y</td>
<td>A short description of the error</td>
</tr>
<tr>
<td>EXCEPTION_FIX</td>
<td>NVARCHAR2</td>
<td>2000</td>
<td>--</td>
<td>Y</td>
<td>A description on how to fix the error</td>
</tr>
<tr>
<td>EXCEPTION_CONTEXT</td>
<td>VARCHAR2</td>
<td>4000</td>
<td>--</td>
<td>Y</td>
<td>The context of the error</td>
</tr>
<tr>
<td>COMPONENT</td>
<td>NUMBER</td>
<td>--</td>
<td>--</td>
<td>N</td>
<td>The BPM component that caused the error</td>
</tr>
<tr>
<td>THREAD_ID</td>
<td>VARCHAR2</td>
<td>200</td>
<td>--</td>
<td>N</td>
<td>The Java thread name in which the error occurred</td>
</tr>
<tr>
<td>STACKTRACE</td>
<td>CLOB</td>
<td>--</td>
<td>--</td>
<td>N</td>
<td>The Java stack trace</td>
</tr>
</tbody>
</table>
Sensor Actions XSD File

The section provides a sample sensor action schema that you can import into Oracle JDeveloper. This schema is also relevant to custom data publishers.

```xml
<?xml version="1.0" encoding="utf-8"?>
<!--
This schema contains the sensor definition. Sensors monitor data and execute callbacks appropriately.
--> 

<!--
Attributes of a sensor action
--> 
<xsd:attributeGroup name="tSensorActionAttributes">
  <xsd:attribute name="name" type="xsd:string" use="optional"/>
  <xsd:attribute name="enabled" type="xsd:boolean" use="optional" default="true"/>
  <xsd:attribute name="filter" type="xsd:string" use="required"/>
  <xsd:attribute name="publishName" type="xsd:string" use="required"/>
  <xsd:attribute name="publishType" type="tns:tSensorActionPublishType" use="required"/>
  <xsd:attribute name="publishTarget" type="xsd:string" use="optional"/>
</xsd:attributeGroup>

<!--
The sensor action type. A sensor action consists:
+ unique name
+ effective date
+ expiration date - Optional. If not defined, the probe is active indefinitely.
--> 
</xsd:schema>
```
+ filter (to potentially suppress data publishing even if a sensor marks it as interesting). - Optional. If not defined, no filter is used.
+ publishName A name of a publisher
+ publishType What to do with the sensor data?
+ publishTarget Name of a JMS Queue/Topic or custom publisher.
+ potentially many sensors.
</xsd:complexType>
<xs:complexType name="tSensorActionList">
    <xsd:sequence>
        <xsd:element name="action" type="tns:tSensorAction" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
</xsd:complexType>
<xs:simpleType name="tSensorKind">
    <xs:restriction base="xsd:string">
        <xs:enumeration value="fault"/>
        <xs:enumeration value="variable"/>
        <xs:enumeration value="activity"/>
    </xs:restriction>
</xs:simpleType>
<xs:complexType name="tActivityConfig">
    <xs:annotation>
        <xs:documentation>The configuration part of an activity sensor comprises of a mandatory 'evalTime' attribute and an optional list of variable configurations</xs:documentation>
    </xs:annotation>
    <xs:complexContent>
        <xs:extension base="tns:tSensorConfig">
            <xs:sequence>
                <xsd:element name="variable" type="tns:tActivityVariableConfig" maxOccurs="unbounded" minOccurs="0"/>
            </xsd:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
<xs:complexType name="tAdapterConfig">
    <xs:annotation>
        <xs:documentation></xs:documentation>
    </xs:annotation>
    <xs:complexType>
        <xs:complexContent>
            <xs:extension base="tns:complexContent">
                <xs:sequence>
                    <xsd:element name="variable" type="tns:ActivityVariableConfig" maxOccurs="unbounded" minOccurs="0"/>
                </xsd:sequence>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
</xs:complexType>
The configuration part of a adapter activity extends the activity configuration with additional attributes for adapters.

```xml
<xs:complexType name="tVariableConfig">
  <xs:complexContent>
    <xs:restriction base="tns:tSensorConfig">
      <xs:attribute name="outputDataType" use="required" type="xsd:string"/>
      <xs:attribute name="outputNamespace" use="required" type="xsd:string"/>
      <xs:attribute name="queryName" use="optional" type="xsd:string"/>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>
```

```xml
<xs:complexType name="tActivityVariableConfig">
  <xs:complexContent>
    <xs:extension base="tns:tVariableConfig">
      <xs:attribute name="target" type="xsd:string" use="required"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

```xml
<xs:complexType name="tFaultConfig">
  <xs:complexContent>
    <xs:extension base="tns:tSensorConfig"/>
  </xs:complexContent>
</xs:complexType>
```

```xml
<xs:complexType name="tNotificationSvcConfig">
  <xs:complexContent>
    <xs:extension base="tns:tActivityConfig">
      <xs:attribute name="inputVariable" use="required" type="xsd:string"/>
      <xs:attribute name="outputVariable" use="required" type="xsd:string"/>
      <xs:attribute name="operation" use="required" type="xsd:string"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

```xml
<xs:complexType name="tSensorConfig">
  <xs:sequence>
    <xs:element name="action" type="tns:tInlineSensorAction" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
```

```xml
<xs:complexType name="tInlineSensorAction">
  <xs:complexContent>
    <xs:restriction base="tns:tSensorAction"/>
  </xs:complexContent>
</xs:complexType>
```
<xsd:complexType name="tSensor">
   <xsd:sequence>
      <xsd:element name="activityConfig" type="tns:tActivityConfig" minOccurs="0"/>
      <xsd:element name="adapterConfig" type="tns:tAdapterConfig" minOccurs="0"/>
      <xsd:element name="faultConfig" type="tns:tFaultConfig" minOccurs="0"/>
      <xsd:element name="notificationConfig" type="tns:tNotificationSvcConfig" minOccurs="0"/>
      <xsd:element name="variableConfig" type="tns:tVariableConfig" minOccurs="0"/>
   </xsd:sequence>
   <xsd:attribute name="sensorName" use="required" type="xsd:string"/>
   <xsd:attribute name="kind" use="required" type="tns:tSensorKind"/>
   <xsd:attribute name="classname" use="required" type="xsd:string"/>
   <xsd:attribute name="target" use="required" type="xsd:string"/>
</xsd:complexType>

<xsd:complexType name="tSensorList">
   <xsd:sequence>
      <xsd:element name="sensor" type="tns:tSensor" minOccurs="0" maxOccurs="unbounded"/>
   </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="tRouterData">
   <xsd:sequence>
      <xsd:element name="header" type="tns:tHeaderInfo"/>
      <xsd:element name="payload" type="tns:tSensorData"/>
   </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="tHeaderInfo">
   <xsd:sequence>
      <xsd:element name="processName" type="xsd:string"/>
      <xsd:element name="processRevision" type="xsd:string"/>
      <xsd:element name="domain" type="xsd:string"/>
      <xsd:element name="instanceId" type="xsd:integer"/>
      <xsd:element name="midTierInstance" type="xsd:string"/>
      <xsd:element name="timestamp" type="xsd:dateTime"/>
      <xsd:element name="sensor" type="tns:tSensor"/>
   </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="tSensorData">
   <xsd:sequence>
      <xsd:element name="activityData" type="tns:tActivityData" minOccurs="0"/>
      <xsd:element name="faultData" type="tns:tFaultData" minOccurs="0"/>
      <xsd:element name="adapterData" minOccurs="0" type="tns:tAdapterData"/>
      <xsd:element name="variableData" type="tns:tVariableData" minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element name="notificationData" type="tns:tNotificationData" minOccurs="0"/>
   </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="tFaultData">
   <xsd:sequence>
      <xsd:element name="activityName" type="xsd:string"/>
      <xsd:element name="activityType" type="xsd:string"/>
   </xsd:sequence>
</xsd:complexType>
<xsd:element name="data" type="xsd:anyType" minOccurs="0"/>
</xsd:complexType>

<xsd:complexType name="tActivityData">
<xsd:sequence>
<xsd:element name="activityType" type="xsd:string"/>
<xsd:element name="evalPoint" type="xsd:string"/>
<xsd:element name="errorMessage" nillable="true" minOccurs="0" type="xsd:string"/>
</xsd:sequence>
</xsd:complexType>

<!-- xml type that will be provided to sensors for variable datas. Note the any element represents variable data. -->
<xsd:complexType name="tVariableData">
<xsd:sequence>
<xsd:element name="target" type="xsd:string"/>
<xsd:element name="queryName" type="xsd:string"/>
<xsd:element name="updaterName" type="xsd:string" minOccurs="1"/>
<xsd:element name="updaterType" type="xsd:string" minOccurs="1"/>
<xsd:element name="data" type="xsd:anyType"/>
<xsd:element name="dataType" type="xsd:integer"/>
</xsd:sequence>
</xsd:complexType>

<xsd:complexType name="tNotificationData">
<xsd:complexContent>
<xsd:extension base="tns:tActivityData">
<xsd:sequence>
<xsd:element name="messageID" type="xsd:string" minOccurs="0" maxOccurs="unbounded"/>
<xsd:element name="fromAddress" type="xsd:string" minOccurs="0"/>
<xsd:element name="toAddress" type="xsd:string" minOccurs="0"/>
<xsd:element name="deliveryChannel" type="xsd:string" minOccurs="0"/>
</xsd:sequence>
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tAdapterData">
<xsd:complexContent>
<xsd:extension base="tns:tActivityData">
<xsd:sequence>
<xsd:element name="endpoint" type="xsd:string"/>
<xsd:element name="direction" type="xsd:string"/>
<xsd:element name="adapterType" type="xsd:string"/>
<xsd:element name="priority" type="xsd:string" minOccurs="0"/>
<xsd:element name="messageSize" type="xsd:string" minOccurs="0"/>
</xsd:sequence>
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>

<!-- The header of the document contains some metadata. -->
<xsd:complexType name="tSensorActionHeader">
<xsd:sequence>
</xsd:complexType>

</xsd:schema>
<xsd:element name="processName" type="xsd:string"/>
<xsd:element name="processVersion" type="xsd:string"/>
<xsd:element name="processID" type="xsd:long"/>
<xsd:element name="midTierInstance" type="xsd:string"/>
</xsd:sequence>
<xsd:attribute name="actionName" use="required" type="xsd:string"/>
</xsd:complexType>

<!--
Sensor Action data is presented in the form of a header and potentially many data elements depending on how many sensors associated to the sensor action marked the data as interesting.
-->
<xsd:complexType name="tSensorActionData">
<xsd:sequence>
<xsd:element name="header" type="tns:tHeaderInfo"/>
<xsd:element name="payload" type="tns:tSensorData" minOccurs="1" maxOccurs="1"/>
</xsd:sequence>
</xsd:complexType>

<!--
Sensor Action data is presented in the form of a header and potentially many data elements depending on how many sensors associated to the sensor action marked the data as interesting.
-->
<xsd:complexType name="tSensorActionData">
<xsd:sequence>
<xsd:element name="header" type="tns:tHeaderInfo"/>
<xsd:element name="payload" type="tns:tSensorData" minOccurs="1" maxOccurs="1"/>
</xsd:sequence>
</xsd:complexType>

<!--
The process sensor value header comprises of a timestamp where the sensor was triggered and the sensor metadata
-->
<xsd:complexType name="tProcessSensorValueHeader">
<xsd:sequence>
<xsd:element name="timestamp" type="xsd:dateTime"/>
<xsd:element ref="tns:sensor"/>
</xsd:sequence>
</xsd:complexType>

<!--
Extend tActivityData to include more elements
-->
<xsd:complexType name="tProcessActivityData">
<xsd:complexContent>
<xsd:extension base="tns:tActivityData">
<xsd:sequence>
<xsd:element name="creationDate" type="xsd:dateTime" minOccurs="0" maxOccurs="1"/>
<xsd:element name="modifyDate" type="xsd:dateTime" minOccurs="0"/>
</xsd:sequence>
</xsd:complexType>

<xsd:simpleType name="tActivityEvalPoint">
<xsd:restriction>
<xsd:enumeration value="start"/>
<xsd:enumeration value="complete"/>
<xsd:enumeration value="fault"/>
<xsd:enumeration value="compensate"/>
<xsd:enumeration value="retry"/>
</xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="tNotificationAction">
<xsd:restriction>
<xsd:enumeration value="creation"/>
<xsd:enumeration value="statusUpdate"/>
</xsd:restriction>
</xsd:simpleType>

<!--
Extend tActivityData to include more elements
-->
<xsd:complexType name="tProcessActivityData">
<xsd:complexContent>
<xsd:extension base="tns:tActivityData">
<xsd:sequence>
<xsd:element name="creationDate" type="xsd:dateTime" minOccurs="0" maxOccurs="1"/>
<xsd:element name="modifyDate" type="xsd:dateTime" minOccurs="0"/>
<xsd:element name="sensor" type="tns:sensor" minOccurs="0" maxOccurs="1"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:complexContent>
</xsd:complexType>

<!-- Extend tVariableData to include more elements -->
<xs:complexType name="tProcessVariableData">
  <xs:complexContent>
    <xs:extension base="tns:tVariableData">
      <xs:sequence>
        <xs:element name="creationDate" type="xsd:dateTime" minOccurs="0" maxOccurs="1"/>
        <xs:element name="modifyDate" type="xsd:dateTime" minOccurs="0" maxOccurs="1"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<!-- Extend tFaultData to include more elements -->
<xs:complexType name="tProcessFaultData">
  <xs:complexContent>
    <xs:extension base="tns:tFaultData">
      <xs:sequence>
        <xs:element name="creationDate" type="xsd:dateTime" minOccurs="0" maxOccurs="1"/>
        <xs:element name="modifyDate" type="xsd:dateTime" minOccurs="0" maxOccurs="1"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<!-- Extend tAdapterData to include more elements -->
<xs:complexType name="tProcessAdapterData">
  <xs:complexContent>
    <xs:extension base="tns:tAdapterData">
      <xs:sequence>
        <xs:element name="creationDate" type="xsd:dateTime" minOccurs="0" maxOccurs="1"/>
        <xs:element name="modifyDate" type="xsd:dateTime" minOccurs="0" maxOccurs="1"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<!-- Extend tNotificationData to include more elements -->
<xs:complexType name="tProcessNotificationData">
  <xs:complexContent>
    <xs:extension base="tns:tNotificationData">
      <xs:sequence>
        <xs:element name="creationDate" type="xsd:dateTime" minOccurs="0" maxOccurs="1"/>
        <xs:element name="modifyDate" type="xsd:dateTime" minOccurs="0" maxOccurs="1"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xsd:complexType name="tProcessNotificationData">
  <xsd:complexContent>
    <xsd:extension base="tns:tNotificationData">
      <xsd:sequence>
        <xsd:element name="creationDate" type="xsd:dateTime" minOccurs="0" maxOccurs="1"/>
        <xsd:element name="modifyDate" type="xsd:dateTime" minOccurs="0" maxOccurs="1"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

Copy of tSensorData type with some modified types.

<xsd:complexType name="tProcessSensorData">
  <xsd:sequence>
    <xsd:element name="activityData" type="tns:tProcessActivityData" minOccurs="0"/>
    <xsd:element name="faultData" type="tns:tProcessFaultData" minOccurs="0"/>
    <xsd:element name="adapterData" minOccurs="0" type="tns:tProcessAdapterData"/>
    <xsd:element name="variableData" type="tns:tProcessVariableData" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element name="notificationData" type="tns:tProcessNotificationData" minOccurs="0"/>
  </xsd:sequence>
</xsd:complexType>

A single process sensor value comprises of the sensor value metadata (sensor and timestamp) and the payload (the value) of the sensor

<xsd:complexType name="tProcessSensorValue">
  <xsd:sequence>
    <xsd:element name="header" type="tns:tProcessSensorValueHeader"/>
    <xsd:element name="payload" type="tns:tProcessSensorData"/>
  </xsd:sequence>
</xsd:complexType>

Process instance header.

<xsd:complexType name="tProcessInstanceInfo">
  <xsd:sequence>
    <xsd:element name="processName" type="xsd:string"/>
    <xsd:element name="processRevision" type="xsd:string"/>
    <xsd:element name="domain" type="xsd:string"/>
    <xsd:element name="instanceId" type="xsd:integer"/>
  </xsd:sequence>
</xsd:complexType>

The list of sensor values comprises of a process header describing the BPEL process with name, cube instance id etc. and a list of sensor values comprising of sensor metadata information and sensor values.

<xsd:complexType name="tProcessSensorValueList">
  <xsd:sequence>
    <xsd:element name="process" type="tns:tProcessInstanceInfo" minOccurs="1"/>
  </xsd:sequence>
</xsd:complexType>
Summary

This chapter describes how to set up and use sensors to monitor BPEL activities, variables, and faults during run time. This chapter also describes how to create sensor actions in Oracle BPEL Process Manager to publish sensor data as data objects in an Oracle BAM Server.
This chapter describes how to build adaptive business processes by using a decision service to integrate BPEL processes with a business rule engine.

This chapter contains the following topics:

- Business Rules and Decision Service Concepts
- Decision Service Architecture
- Use Cases for Integration of Business Processes and Business Rules
- Integration of BPEL Processes with Business Rules
- Methodology for Rule Set Modeling and Integration with a BPEL Process
- Decision Service Deployment and Run Time
- Advanced Decision Service Features
- Example of BPEL Process Integration with Business Rules

See Also: The AutoLoanDemo tutorial, which describes how to design a BPEL process that integrates with business rules and uses human workflow:

`SOA_Oracle_Home\bpe\samples\demos\AutoLoanDemo`

### Business Rules and Decision Service Concepts

This section provides an overview of Oracle BPEL Process Manager support for business rules and business rule engines.

This section contains the following topics:

- Business Rules and Business Rule Engines
- Decision Service
- Oracle Business Rules with Oracle BPEL Process Manager

### Business Rules and Business Rule Engines

Business rules are statements that describe the policies of a company. Examples of business rules can include the following:

- All customers that buy more than $100 worth of products at one time or who are over the age of 65 receive a 10% discount.
Business Rules and Decision Service Concepts

- A sales department is notified when inventory quantity is lower than ten and there are more than five pending orders on a given day.
- If the annual income of a customer is less than $10,000, a loan request is denied.
- If a customer submitted a late payment for a previous purchase, an additional charge of 2% is added to their next purchase.

A business rule engine is a system that manages and executes business rules. A business rule system typically consists of a rule repository, rule author, and rule engine. The rule author allows business rules to be specified separately from application code. Separating the business rules from code allows business analysts to change business policies quickly with graphical tools. The rule engine evaluates the business rule and returns decisions or facts that are then used in the business process. The rules are typically stored in a rule repository in a database or file system.

Decision Service

Oracle BPEL Process Manager provides support for a decision service. A decision service is a mechanism for publishing rules and rule sets as a reusable service that can be invoked from multiple business processes. The decision service is a standalone deployment unit that consists of the following components:

- A Web service that wraps the rule session to the underlying rule engine. This service lets business processes assert and retract facts as part of the process. In some cases, all facts can be asserted from the business process as one unit. In other cases, the business process can incrementally assert facts and eventually consult the rule engine for inferences. Therefore, the service has to support both stateless and stateful interactions.
- Rules that are evaluated by the decision service using the rule engine. These are defined using the rule author and loaded into the rule repository.
- Metadata that describes facts required for specific rules to be evaluated. Each rule exposed as a service uses different types of facts. These facts must be exposed through XSD definitions. Appropriate WSDL operations must be exposed for rule evaluation.

For example, a credit rating rule set may expect a customer’s SSN, previous loan history, and so on as facts, but a pension payment rule may expect an employee’s years of service, salary, age, and so on as facts.

See Also: "Decision Service Architecture" on page 18-3

Oracle Business Rules with Oracle BPEL Process Manager

Oracle BPEL Process Manager provides support for Oracle Business Rules. Oracle Business Rules is a component included with Oracle Application Server. Business analysts use Oracle Business Rules to create and change business rules that are separate from the application code. This enables analysts to change business rules without stopping business processes or having to involve programmers.

In Oracle Business Rules, facts are data objects asserted in the Oracle Business Rules Rules Engine. For example:

- For a car rental company to create a rule to match a driver’s age, the driver’s age represents the fact used in the rule.
- For a loan company to create a rule denying a loan request to customers whose incomes fall below a specific level, the income level represents the fact used in the rule.
Each data object instance corresponds to a single fact. Rules are expressions that can be evaluated on this factual information.

If an object is re-asserted (whether or not it has changed), the Oracle Business Rules Rules Engine is updated to reflect the new state of the object. Re-asserting the object does not create a new fact. In order to have multiple facts of a particular fact type, separate object instances must be asserted.

Using the Oracle Business Rules Rule Author, you create rules that operate on facts that are part of a data model. You make business objects and their methods known to Oracle Business Rules using fact definitions.

Oracle Business Rules consist of the following key components:

- **Oracle Business Rules Rule Author** — A Web browser-based tool that provides a point-and-click interface to create and edit rules.

- **Oracle Business Rules Rules Engine** — A Java library that applies rules to facts and defines and processes rules. The Oracle Business Rules Rules Engine defines a declarative rule language, provides a language processing engine, and provides debugging tools.

Oracle BPEL Process Manager provides support for the following Oracle Business Rules Rules Engine repositories that store the business rules:

- Oracle Rules Engine File Repository — Stores rules in a file repository.
- Oracle Rules Engine WebDav Repository — Stores rules in a Web Distributed Authoring and Versioning (WebDAV) repository.

A repository stores dictionaries. A dictionary is a set of XML files that stores the application’s rule sets and the data model. Rule sets are a group of business rules that are typically evaluated together and generated as one unit. Dictionaries can have different versions. Dictionaries and dictionary versions can be created, deleted, exported, and imported into a repository.

- **Oracle Business Rules SDK** — A Java library that provides business rule management features to use for writing customized rules programs.


**See Also:**

The following documentation:

- Oracle Business Rules User’s Guide
- Oracle Business Rules Java API Reference
- Oracle Business Rules Language Reference Guide

**Decision Service Architecture**

This section describes the components that comprise the decision service.

This section contains the following topics:

- Decision Service Components
- Interaction with Other Components
Contents of Decision Service Configuration File

Decision Service Components

The decision service consists of the following components:

- Rule Provider Interface (RPI) — An interface used by decision service design time clients such as Oracle JDeveloper. The RPI hides the details of a concrete rule engine implementation. This enables the RPI to interface with any rule engine from any provider. The main purpose of the RPI is to expose a uniform view of rule engine metadata such as fact types, rule sets, dictionaries, and so on.

  A design time component (such as Oracle JDeveloper) can use the RPI to browse the metadata of a backend rule engine. According to what you model, metadata information for a decision service partner link can be materialized in the decision service configuration file.

- Decision Service Builder — Reads the metadata information from a decision service configuration file and creates a self-contained J2EE enterprise archive that can be deployed to an application server.

- Decision Service Runtime — A JAX-RPC Web service that is the Web service enabler for business rule engines such as Oracle Business Rules. The runtime itself consists of the following components:

  - Decision Service Cache — Maintains metadata information about the rule data model used by the service. This includes metadata about the fact types, rule set, and decision service configuration. In addition, all stateful rule sessions are stored in that cache. Oracle Java Object Cache (JOC) is used. Therefore, the cache can be configured to run in clustered environments.

  - Fact Converter — Converts data coming from and going to Oracle BPEL Process Manager to a format understood by a rule engine. For the Oracle Business Rules Rules Engine, the fact converters use the JAXB 1.0 tech stack to convert BPEL variable data (XML) to and from Java objects.

  - Execution Unit — Executes the various steps defined by the interaction pattern at the backend rule engine. The execution unit uses the RPI for executions.

See Also: Oracle Containers for J2EE Services Guide for cache configuration options to use in clustered environments

Interaction with Other Components

Figure 18–1 shows how the decision service interacts with other components.
1. The rule author is used for rule modeling.
   The rule model is saved to a rule repository. The rule repository can be a file or a directory at a WebDav backend.

2. The RPI of the decision service is used by the Decision Service wizard to create a connection to the rule repository and browse the repository metadata.

3. The Decision Service wizard creates the decision service metadata configuration file. The metadata consists of information about the backend rule engine being used (rule provider) and the interaction patterns (together with fact type information) modeled for the partner link.

4. The decision service builder creates the decision service enterprise archive for deployment into an application server. As part of this, a WSDL is created with message types and operations adjusted to what you modeled in Step 3.

5. The decide activity uses decision service metadata information to present you with a list of available operations of the service and detailed information on the number and types of facts used for an interaction with the rule engine.

6. The decide activity completes generation of a new BPEL scope in the BPEL process model. Appropriate assign activities are created to convert the data from BPEL variables to data that the decision service (and more importantly the backend rule engine) expects.

7. The BPEL process is deployed to Oracle BPEL Process Manager during deployment time.

8. The decision service enterprise archive is deployed to the application server.

9. The BPEL process instance invokes the JAX-RPC decision service at run time, which then interacts with the backend rule engine, executes rule sets, invokes functions, and so on. Results are eventually returned to the BPEL process.

**Contents of Decision Service Configuration File**

The decision service configuration file (*decisionservices.xml*) defines the contract between the various components involved in the interaction of the decision service with the design time and a backend rule engine.

The decision service configuration file consists of two parts:
The first part specifies metadata about the rule engine connections.

The second part provides the metadata for specific interaction patterns with a backend rule engine.

For example:

```xml
<decisionServices xmlns="http://xmlns.oracle.com/bpel/rules">
  <ruleEngineProvider provider="Oracle" name="CreditRatingRuleRepository">
    <repository type="File">
      <file>/D:/bpeldev/10.1.3/demo/LoanDemoPlusRules/repository/CreditRatingRepository</file>
    </repository>
  </ruleEngineProvider>
  <ruleEngineProvider provider="Oracle" name="LoanAdvisorRepository">
    <repository type="File">
      <file>/D:/bpeldev/10.1.3/demo/LoanDemoPlusRules/repository/CarLoanBrokerRepository</file>
    </repository>
  </ruleEngineProvider>
  <decisionService name="CreditRatingAgent" targetNamespace="http://cr.org/CreditRatingAgent" ruleEngineProviderReference="CreditRatingRuleRepository">
    <catalog>RatingFY06</catalog>
    <catalogVersion>Approved_060205</catalogVersion>
    <ruleset>PrivateCustomerRatingRules</ruleset>
    <pattern name="AssertExecuteWatchStateless">
      <arguments>
        <assert>creditrating.Ratingrequest</assert>
        <watch>creditrating.Rating</watch>
      </arguments>
    </pattern>
  </decisionService>
  <decisionService name="LoanAdvisorAgent" targetNamespace="http://laa.org/LoanAdvisorAgent" ruleEngineProviderReference="LoanAdvisorRepository">
    <catalog>LoanOfferings</catalog>
    <catalogVersion>REVIEWED_060518</catalogVersion>
    <ruleset>PrivateCustomerRatingRules</ruleset>
    <pattern name="CallFunctionStateless">
      <arguments>
        <call>DM.computeAdvisePrivateFinancing</call>
      </arguments>
    </pattern>
  </decisionService>
</decisionServices>
```

The configuration file includes the following elements:

- **ruleEngineProvider** — Specifies metadata about a backend rule engine connection. Apart from the rule engine provider, information on the rule repository is specified. You distinguish between these types of repositories:
  - File — The rule repository is stored in a file or directory.
  - WebDav — The rule repository is stored in a WebDav location.

- **decisionService** — Consists of a name, an optional target namespace, and a mandatory reference to the rule engine to use for the interaction. A complete interaction with the rule engine is specified, which includes the catalog and catalog version to use, and the rule set to execute. Various interaction patterns are
Integration of BPEL Processes with Business Rules

Use Cases for Integration of Business Processes and Business Rules

Oracle BPEL Process Manager and business rules are complementary technologies. Oracle BPEL Process Manager focuses on orchestration of systems, services, and people. Business rules focus on decision making and policies.

Some examples of where decision service can be used include:

- Dynamic processing — Rules can perform intelligent routing within the business process based on service level agreements or other guidelines. For example, if the customer requires response within one day, send the loan application to the StarLoan loan agency only. If the customer can wait longer, then route the request to three different loan agencies.

- Externalize decision points in the process — There are typically many conditions that must be evaluated as part of a business process. However, the parameters to these can be changed independent of the process. For example, you provide loans only to customers with a credit score of at least 650. This value may be changed dynamically based on new guidelines set by business analysts.

- Data validation and constraint checks — Rules can validate input documents or apply additional constraints on requests. For example, you provide loans only to customers with a credit score of at least 650. This value may be changed dynamically based on new guidelines set by business analysts.

- Human workflow — Rules are frequently used in the context of human tasks in the business process:
  - Policy-based task assignments dispatch tasks to specific roles or users. For example, a process that handles incoming requests from a portal can route loan requests and insurance quotes to a different set of roles.
  - Load balancing of tasks among users — When a task is assigned to a set of users or a role, each user in that role acquires a set of tasks and acts on them in a specified time. For new incoming tasks, policies may be applied to set priorities on the task and put them in specific user queues. For example, a specific loan agent is assigned a maximum of 10 loans at any time.

Integration of BPEL Processes with Business Rules

Oracle BPEL Process Manager provides the following design-time components that enable you to integrate BPEL processes with business rules.

- Create Rule Engine Connection Wizard
- Decision Service Wizard
- Decide Activity

See Also: “Example of BPEL Process Integration with Business Rules” on page 18-51

Create Rule Engine Connection Wizard

The Create Rule Engine Connection wizard enables you to create a connection to a rule engine. This connection enables you to browse and select rule sets and functions available in rule dictionaries of a rule repository in the business rule engine. You only
need to create one connection to the rule engine. Once created, this connection is shared between multiple BPEL projects.

1. Select Connection Navigator from the View main menu in Oracle JDeveloper.
2. Right-click Rule Engines and select New Rule Engine Connection.
3. Click Next on the Welcome window.
4. Enter a name. When creation of the rule engine connection is complete, this name displays in the Connection Navigator under Rule Engines.
5. Select the type of repository in which the rule sets and functions are stored in the repository of the business rule engine. For this release, the Oracle Business Rules Rules Engine is supported.
6. Click Next.
7. Click the folder icon to select a file repository directory. If you selected Oracle Rules Engine WebDav Repository, you are instead prompted to select a WebDav connection to the repository.
Integration of BPEL Processes with Business Rules

8. Click Next.
   The Test Connection window appears.

9. Click Test.
   If the connection to the business rule repository is successful, the following
   message appears:
   Success
   If the connection is unsuccessful, click Details to view the reason for failure. Take
   appropriate corrective actions.

10. Click Finish.
    The connection name displays in the Connection Navigator under Rule Engines.
    If you need to edit the connection, double-click the connection name to display
    configuration details.

**Decision Service Wizard**

The Decision Service wizard enables you to integrate your BPEL process with a
business rule (for example, a rule set or function) that you created in the Oracle
Business Rules Rules Engine. This enables you to make business decisions based on
these rules.

Figure 18–3 provides an overview of this integration process.

**Figure 18–2 Decision Service**

This wizard performs the following tasks:
- Guides you through the selection of a rule set or function from a repository and
  the invocation pattern (operation) to perform
- Converts your selection into a Web service to use in the BPEL process
A new decision service partner link for this Web service is automatically created that
interfaces with the Oracle Business Rules Rules Engine. A WSDL file based on the rule
set is generated.

The Decision Service wizard provides the following benefits:

---

**Note:** You can create a WebDav connection by right-clicking
WebDAV Server and selecting New WebDAV Connection in the
Connection Navigator.
Integration of BPEL Processes with Business Rules

- Dynamic processing (provides for intelligent routing, validation of policies within a process, and constraint checks)
- Integration with adhoc workflows (provides policy-based task assignment, various escalation policies, and load balancing of tasks)
- Integration with business activity monitoring (sends alerts based on certain policies and dynamic processing-based key performance indicator (KPI) reasoning)

The following sections describe Decision Service wizard functionality in further detail:

- Selecting an Invocation Pattern
- Selecting a Business Rule
- Specifying Input and Output Facts
- Importing Schema Files

Selecting an Invocation Pattern

The Decision Service wizard enables you to select an invocation pattern that describes how to interact with the business rule engine.

1. Drag and drop a Decision Service from the Services list of the Component Palette onto either side of the designer window beneath the header Services.
2. Enter a name in the Service Name field. When complete, this name is used as the partner link name. A WSDL file of the same name is also created. The Namespace field is automatically completed with your entry from the Service Name field. You must have a unique namespace for each WSDL in your project. This is because some BPEL variables are generated from elements in the WSDL files.
3. Select an invocation pattern for invoking the rule set or function. The wizard creates rule sessions based on the invocation pattern you select. You do not have the option of selecting between stateful or stateless rule sessions.

The following invocation patterns are available for selection:

- Execute Ruleset

18-10 Oracle BPEL Process Manager Developer's Guide
Execute function

4. Click the flashlight next to the Ruleset or Function field. The name of this field is based on your selection in the Invocation Pattern field. This opens the Explorer window.

Selecting a Business Rule

1. Click Show All Versions to display all the catalog versions of rule dictionaries in the rule repository of the business rule engine.

2. Expand and select the dictionary (RatingFY06), dictionary version (Approved_06012005), and rule set (PrivateCustomerRatingRules).

3. Click OK. You are returned to the Select a Ruleset or a Function window.

Notes:

- If you have not created a connection to the rule engine, you cannot access the rule sets and functions in the rule repository. Click the first icon in the upper right to start the Create Rule Engine Connection wizard.

- An alternative to creating a connection to a file-based rule repository is to directly import a file into a project. Note that each BPEL project must physically copy the repository file to the project at the time of Web service creation. After the file is copied, any changes to the original file are not reflected in the Web service. Click the second button in the upper right to browse for the file to import. The file must be a valid Oracle rule repository file. Otherwise, an error appears.

3. Click OK.

You are returned to the Select a Ruleset or a Function window.
Specifying Input and Output Facts

Based on the type of invocation pattern you selected, the Select a Ruleset or a Function window displays a table with different details at the bottom of the window:

- **Rule Sets**
- **Functions**

### Rule Sets

The table displays the facts in the data model of the catalog from which you selected the rule set. You select the facts to assert (that is, set the value from the BPEL variable to the fact) and retrieve the results to return. Note that the columns that appear are based on whether the selected pattern asserts facts, retrieves facts, or does both.

1. **Specify the input (assert) and (optionally) the output (watch) facts.** The assert facts enable you to assert a fact to the rule set or function (send factual data to the business rule engine). The watch facts enable you to return results from the rule set or function. Watch facts only appear if you selected an invocation pattern that retrieves results.

   The **Check here to assert all descendants from the top level element** check box enables you to assert all the descendants of the selected rule set or function. For example, assume that a purchase order rule set contains three fact types. If this check box is selected, the purchase order and all three fact types are asserted. If this check box is not selected, only the purchase order is asserted.

2. **Click Next.**

### Functions

The table displays the input parameters and parameter types to return.
1. Click Next.

Importing Schema Files

1. Review the on-screen messages to ensure that all necessary XSD schema files for this project are imported from the repository by the wizard.

The wizard attempts to identify all the schema files in the repository that must be imported into this project. Based on this attempt, this window can display the following status messages:

- If the Decision Service Wizard finds the schema files to import, the directory paths to the files display at the top of this window. No action is required on your part.
- If the Decision Service Wizard cannot find the schema files to import, the directory paths to the files display at the top of this window. You must manually copy these files to the specified directory.
- If this XSD schema file includes or imports other schema files, ensure that these files are copied into the `bpel\rules\xsd` subdirectory of your BPEL.
project indicated on-screen. Ensure that you use only relative directory paths for these schema files.

2. Click Next.

The decision service partner link is created. A directory named decisionservices is also created in the BPEL project. A directory with the same name as the service name is created inside the decisionservices directory.

See Also: "Decision Service Partner Link Directory Structure" on page 18-39 for specific details about directories and files created with the decision service partner link

**Decide Activity**

The decide activity enables you to create a BPEL process activity that invokes the decision service partner link you created with the Decision Service wizard. This activity also enables you to create copy operation assignments between the fact data in your rule set or function and BPEL variables.

When complete, a decide activity consisting of assign and invoke activities to the decision service partner link is created.

Figure 18–3 provides an overview of this integration process.

**Mapping Input and Output Facts to BPEL Variables**

1. Drag and drop a Decide activity from the Process Activities list of the Component Palette into your BPEL process.
2. Enter a name.
3. Select the decision service partner link you created. If you have not created a
decision service, click the first icon to the right of the **Decision Service** field.

4. Select the operation of the invocation pattern to perform. The operations available
for selection are based on the invocation pattern you selected in Step 3 on
page 18-10.

   - If you selected **Execute Ruleset**
     - **Assert facts only** — Select the rule engine facts you want to assert (send
       factual data to the rule engine) in the future. You assign the required data
       for the facts with a BPEL assign activity. The underlying rule session must
       be stateful. Otherwise, the asserted facts are not visible to subsequent
       rule engine invocations.
     - **Retrieve results** — Retrieve a result from the business rule engine. The
       values of these results may have changed by past execution of a rule set
       acting on these facts. The wizard assumes that it has a stateful rule session
       in its cache from which a result can be retrieved. This is the case if the
       invocation pattern **Assert facts and execute rule set** operation was
       executed before in the BPEL process.
     - **Assert facts and execute rule set** — The same as **Assert facts only**, except
       that the rule set is executed after the facts are asserted. The wizard creates
       (or uses) a stateful rule session. Otherwise, the result of executing this
       pattern is lost. No results are retrieved from the business rule engine.
     - **Assert facts, execute rule set, and retrieve results** — The same as **Assert
       facts and execute rule set**, except that the results are retrieved from the
       business rule engine. You map the results of rule set execution to BPEL
       variables with an assign activity. The rules session remains active. This
       enables you to reuse previously asserted facts.
     - **Assert facts, execute rule set, retrieve results, and reset the session** —
       The same as **Assert facts, execute rule set, and retrieve results**, except that
       the results are reset for the next time that you invoke the Web service.
       Resetting the session clears the previously asserted fact values.

   - If you selected **Execute function**
     - **Execute function** — Executes a function. Functions are also defined in
       dictionaries. For rule sets, you select input and output facts. For functions,
       there are a fixed set of input parameters and a single return value.
     - **Execute function and reset the session** — The same as **Execute function**, except
       that a stateful rule session is created for this pattern. All fact values
       are reset after retrieving the return value of the function.

5. Click **Assign Input Facts**, then click **Create** to create mappings for the input facts.
This enables you to assign BPEL variables to the facts to be asserted or to the
function input parameters.
This enables you to create assignments that map BPEL input variables to automatically created BPEL variables that correspond to the input (assert) fact type shown in “Specifying Input and Output Facts” on page 18-12 (for this example, Rating request).

6. If you selected an invocation pattern that retrieves results, click Assign Output Facts, then click Create to create mappings for the output facts. This enables you to assign values from a function return value or rule set result to a BPEL variable.

This enables you to create assignments that map automatically created BPEL variables that correspond to the output (watch) fact type shown in “Specifying Input and Output Facts” on page 18-12 (for this example, Rating) to BPEL input variables.

7. Click OK when complete.

A decide activity consisting of assign and invoke activities to the decision service partner link is created.
Methodology for Rule Set Modeling and Integration with a BPEL Process

Rule sets are a group of business rules that are typically evaluated together and generated as a single unit. This section describes two methodologies for modeling rule sets in a rule author.

After you model a rule set, you can integrate it with a BPEL process in Oracle JDeveloper. You must have an existing rule repository prior to creating a decision service partner link in Oracle JDeveloper.

This section contains the following topics:

- **Recommended Methodology**
- **Methodology One: Modeling Fact Types Based on an XML Schema**
- **Methodology Two: Modeling Rules Based on Existing RL or JavaBeans Fact Types**
- **Invoking the Sample Rule Set from a BPEL Process**
- **Summary of Methodology**

**Recommended Methodology**

Oracle recommends that you follow these steps when modeling rule sets in a rule author.

- Create a data model for rule authoring based on the XML schema.
- Create a new rule repository and dictionary in the rule author.
- Import the XML schema into the data model of the rule dictionary as XML facts.
- Create a new rule set and model rules.
- Use the Decision Service wizard to create a partner link.

**Methodology One: Modeling Fact Types Based on an XML Schema**

This section describes how to model a simple rule for a credit rating:

- **Task 1: Create a Data Model for Rule Authoring**
- **Task 2: Create a New Rule Repository and Dictionary in the Rule Author**
Methodology for Rule Set Modeling and Integration with a BPEL Process

- Task 3: Import the XML Schema into the Data Model of the Rule Dictionary
- Task 4: Create a New Rule Set and Model Rules

After completion, you use the Decision Service wizard to integrate your rule set with a BPEL process.

See Also: "Invoking the Sample Rule Set from a BPEL Process" on page 18-33

Task 1: Create a Data Model for Rule Authoring

The first step in rule modeling is to define a data model. Data models based on XML schema are supported. This example begins with a simple data model for credit rating and defines two elements:

- `ratingRequest`

  ```xml
  <xsd:element name="ratingRequest">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="date" type="xsd:date" maxOccurs="1"/>
        <xsd:element name="SSN" type="xsd:string" maxOccurs="1"/>
        <xsd:element name="name" type="xsd:string" maxOccurs="1"/>
        <xsd:element name="age" type="xsd:int" maxOccurs="1"/>
        <xsd:element name="amount" type="xsd:double" maxOccurs="1"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  ```

- `rating`

  ```xml
  <xsd:element name="rating">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="date" type="xsd:date" maxOccurs="1"/>
        <xsd:element name="SSN" type="xsd:string" maxOccurs="1"/>
        <xsd:element name="rating" type="xsd:int" maxOccurs="1"/>
        <xsd:element name="risk" type="xsd:string" maxOccurs="1"/>
        <xsd:element name="maxAmount" type="xsd:double" maxOccurs="1"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  ```

Note: It is important to understand that rule modeling is performed at the XML schema type level, whereas the data transfer from BPEL to the decision service is at the element level.

An alternative model (with the same semantics) is to define the following XML schema:

```xml
<xsd:element name="ratingRequest" type="tRatingRequest"/>
<xsd:element name="rating" type="tRating"/>
<xsd:complexType name="tRatingRequest">
  <xsd:sequence>
    <xsd:element name="date" type="xsd:date" maxOccurs="1"/>
    <xsd:element name="SSN" type="xsd:string" maxOccurs="1"/>
    <xsd:element name="rating" type="xsd:int" maxOccurs="1"/>
    <xsd:element name="risk" type="xsd:string" maxOccurs="1"/>
    <xsd:element name="maxAmount" type="xsd:double" maxOccurs="1"/>
  </xsd:sequence>
</xsd:complexType>
```
Methodology for Rule Set Modeling and Integration with a BPEL Process

<xsd:element name="amount" type="xsd:double" maxOccurs="1"/>
</xsd:complexType>

<xsd:complexType name="tRating">
<xsd:sequence>
<xsd:element name="date" type="xsd:date" maxOccurs="1"/>
<xsd:element name="SSN" type="xsd:string" maxOccurs="1"/>
<xsd:element name="rating" type="xsd:int" maxOccurs="1"/>
<xsd:element name="risk" type="xsd:string" maxOccurs="1"/>
<xsd:element name="maxAmount" type="xsd:double" maxOccurs="1"/>
</xsd:sequence>
</xsd:complexType>

1. Open a text editor.
2. Save either schema to a file named CreditRatingTypes.xsd. Ensure that you add the appropriate opening and closing header and footer information to the schema file.

**Task 2: Create a New Rule Repository and Dictionary in the Rule Author**

You now create a new rule repository and dictionary in the rule author.

1. Copy the SOA_Oracle_Home\rules\fileRepositories\ruleRepository file to a location within your file system. For example:
   C:\CreditRatingRules\model\CreditRatingRepository
   
   You are now ready to open a new repository in the rule author, log in to the rule author, and connect to the new repository.

2. Log in to the rule author and connect to the new repository.
   http://hostname:8888/ruleauthor

3. Enter oc4jadmin/password.
   where password is the password you specified for the oc4jadmin user during installation.

4. Click Repository.
   The Connect page appears.

5. Select File from the Repository Type list.
6. Click Browse to the right of the File Location field.
7. Select the repository file location specified in Step 1.
8. Click Create.
   A message indicates that a repository connection has been created.
9. Click the Create subtab.
   The Create Dictionary page appears.
10. Enter SampleDictionary in the New Dictionary Name field and click Create.
A message indicates that the dictionary has been imported.

You now import the XML schema data model into the rule dictionary.

Task 3: Import the XML Schema into the Data Model of the Rule Dictionary

For a given rule dictionary, you can define the underlying data model to use for rule authoring.

1. Click the Definitions tab.
2. Click XMLFact in the Definitions tree on the left.
   The XMLFact Summary page appears.
3. Click Create.
   The XML Schema Selector page appears.

4. Enter the following details:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML Schema</td>
<td>Enter the absolute path to the CreditRatingTypes.xsd file you created in Step 2 of &quot;Task 1: Create a Data Model for Rule Authoring&quot; on page 18-19. For this example, the second schema file model described in that step was used.</td>
</tr>
<tr>
<td>JAXB Class Dictionary</td>
<td>Enter a directory in which to create the JAXB classes.</td>
</tr>
<tr>
<td>Target Package Name</td>
<td>Enter a Java package name to use for the XML fact types (for example, creditrating).</td>
</tr>
</tbody>
</table>

5. Click Add Schema.
6. Expand the Generated JAXB Classes tree at the bottom.
7. Specify the objects in the schema to import. For this example, RatingRequest, Rating, TRatingRequest, and TRating are selected.

8. Click Import.
   A confirmation message displays indicating that four classes or packages have been imported.
   You can now specify meaningful aliases for your data model. This action is optional, but considered a good practice for rule modeling.

9. Click XMLFact in the Definitions tree.
10. Select creditrating.TRatingRequest from the XML Fact Summary table.
11. Click Edit.
   The Properties section appears.
12. Enter appropriate text in the Alias column for the SSN, age, amount, date, and name properties.

12. Click OK.
Notes: Oracle Business Rules use Oracle JAXB 1.0 for XML fact types. As a result, certain limitations apply:

- Use different names for elements and complex types. Although the XML schema specification allows the same name for an element and a type, the JAXB class generator does not support it.

- Use different target package names for every XML schema imported into the rule author. As part of JAXB class generation, a factory class ObjectFactory is created. If you import a second XML schema and specify the same target package name, the JAX generator overwrites the factory class from the first import. This results in unexpected behavior from the rule engine.

As a result of using JAXB for fact types, rule modeling needs to happen on the XML schema type level (complexType level). This is because for XML elements, JAXB generates marker interfaces only and the rule author cannot introspect the attributes and methods of these interfaces for rule modeling.

14. Click RLFunction in the Definitions tree.
   The RLFunction Summary page appears.

15. Click Create.
   The RLFunction page appears.

16. Enter the following details:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>DM assertRating</td>
</tr>
<tr>
<td>Alias</td>
<td>assertRating</td>
</tr>
<tr>
<td>Return Type</td>
<td>void</td>
</tr>
</tbody>
</table>

The page looks as follows:
17. Enter the following function in the Function Body section:

```java
// Create a new Rating object using JAXB ObjectFactory
creditrating.ObjectFactory of = new creditrating.ObjectFactory();
creditrating.Rating rating = of.createRating();
// Get a calendar instance for the current date
java.util.Calendar calendar = java.util.Calendar.getInstance();
// Set Rating object attributes
rating.setSSN(req.getSSN());
rating.setDate(calendar);
rating.setRating(cr);
rating.setRisk(crisk);
rating.setMaxAmount(cmax);
// Assert Rating object in working memory
assert (rating);
```

18. Click Apply.

The following confirmation message appears:

This entity has been updated successfully.

**Task 4: Create a New Rule Set and Model Rules**

After the data model is defined, you can create the rules for the credit rating.

1. Click the Rulesets tab.
2. Click Create in the RuleSet Summary page.

The Ruleset page appears.

3. Enter SampleRuleset in the Name field.
4. Enter an optional description in the Description field.
5. Click OK.
7. Click Create in the Rules section.
8. Enter the following details:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>YoungCustomers</td>
</tr>
<tr>
<td>Description</td>
<td>Rule for young customers</td>
</tr>
<tr>
<td>Priority</td>
<td>0</td>
</tr>
</tbody>
</table>

When complete, the Rule page appears as follows:

9. Click New Pattern to define the If pattern of the rule (the rule matching part).

The Pattern Definition page appears asking for the pattern of the young customer rule.

10. Enter an optional name for the expected fact to match in the Choose Pattern section.

11. Select TRatingRequest from the drop-down list.

It now becomes clear what it means to model the rules on the XML schema type level, and not on the element level.

12. Click Create in the Define Test for Pattern section.

13. Select the appropriate operands from the Field column.
14. Specify several matching statements for the rule. For this example, the young customer rule is executed if the following statements are satisfied:

- If the rating request comes from a customer at least 18 years of age and less than 35 years of age.
- If the requested loan amount is less than 20,000.

15. Click **Apply**.

16. Specify the **Then** section of the rule.
   As part of the action, you want to retract (remove) the original rating request object from the working memory of the rule engine and create and assert a new rating object.

17. Click **New Action** in the **Then** section of the Rule page to retract the original request object (the object that caused the rule to be invoked).

18. Select **Retract** from the **Action Type** list.

19. Select **request** from the **Fact Instance** list.
20. Click **OK**.

21. Click New Action again and select **Call** from the Action Type list.

22. Select **assertRating** from the Function list. This is the function created in Step 16 on page 18-22.

23. Provide values for the function parameters. For this example, **ratingrequest**, **credit_rating**, **credit_risk**, and **credit_max_amount**. The values for **ratingrequest** and **credit_max_amount** cause the rule to be invoked and can be used in the rule action part.

24. Click **OK**.

The rule appears as follows:
25. Double-check the rule and confirm by clicking **OK**.

   It is a good practice to model a rule with no \texttt{if} statement pattern (simply accepting the fact) and an action that generates a well-defined result for the case where no other rule is invoked. For example:
   
   - Set the priority to \texttt{-10} in the \texttt{Priority} field to ensure that this rule is not invoked if other rules match the pattern of the fact instance.
   - Accept the fact (request is a \texttt{TRatingRequest} in the \texttt{If} section) without specifying any additional test pattern.
   - Provide a dummy result that can be checked later from Oracle BPEL Process Manager (Call \texttt{assertRating(request, 0, "Unknown", 0.0)} in the \texttt{Then} section).

26. Click **Save Dictionary** in the upper right corner.

27. Enter a name for the dictionary when prompted and click **Save**.

   The rule set modeling process is now complete. The rules can now be used in a BPEL process.
Methodology Two: Modeling Rules Based on Existing RL or JavaBeans Fact Types

In "Methodology One: Modeling Fact Types Based on an XML Schema", you modeled fact types based on an XML schema for integration with Oracle BPEL Process Manager. This methodology describes the case in which the rules are modeled already based on RL or JavaBean fact types.

The following methodology is taken:

■ Model the contract between BPEL and the business rules using an XML schema.
■ Create an RL function in the rule author that accepts parameters of the modeled XML schema. Then, perform the following procedures:
  – Convert the parameter values to RL or Java fact type objects.
  – Execute the rule set in question (rules are modeled on top of the RL or Java data model).
  – Convert the resulting fact object (RL or Java) to an object of the RL function return type.
■ Use the Call Function pattern (the invocation pattern you can select) from Oracle JDeveloper.

This section describes how to model a rule set in which the rules are already modeled based on RL or JavaBean fact types:

■ Task 1: Define a Contract between BPEL and Business Rules
■ Task 2: Create a New Data Model Using the RL Fact Types
■ Task 3: Create a New Rule Set and Rules
■ Task 4: Create the RL Function Contract

Task 1: Define a Contract between BPEL and Business Rules
You must define a contract between the process modeled in BPEL and the business rules. Assume the contract is defined as follows:

■ Input — A rating request document of element ratingrequest
■ Output — A rating document of element rating
■ Rule set — Execute rules for the credit rating

The contract can be expressed in terms of an RL function calculateCreditRating with the following signature:

creditrating.Rating calculateCreditRating(creditrating.Rating request)

Before creating the function in the rule author, you define a new data model based on RL fact types and a new rule set based on the RL fact type data model.

Task 2: Create a New Data Model Using the RL Fact Types
You create a new data model for the credit rating using RL fact types.

1. Load the sample dictionary from the "Methodology One: Modeling Fact Types Based on an XML Schema” into the rule author.
2. Click the Definitions tab.

![Diagram of RLFact with name MyRatingRequest]

5. Create a new RL fact type named MyRating.

![Diagram of RLFact with name MyRating]

6. Click OK.
   
   For the function to work properly, you must create a global variable to use as a placeholder to carry the result of rule execution.

7. Select Variable in the Definitions tree.
8. Select Create.
9. Create a variable named theResult of type mr.
   
   This is the alias of RL fact type MyRating.
Task 3: Create a New Rule Set and Rules

1. Click the Rulesets tab.
2. Click Create in the RuleSet Summary page.
3. Enter AlternateRuleset in the Name field.
4. Enter an optional description in the Description field.
5. Click OK.
7. Click Create in the Rules section.
8. Click New Pattern to enter the If pattern of the rule (the rule matching part).
9. Specify several matching statements for the rule. For this example:
   - If the rating request comes from a customer at least 18 years of age and less than 35 years of age.
   - If the requested loan amount is less than 20,000.
As part of the action, you want to retract (remove) the original request object.

10. Click OK.
11. Click New Action in the Then section.
12. Select Retract from the Action Type list.
13. Select req from the Fact Instance list.

Add Action

Choose an action.

Action Type: Retract

- Define parameters associated with the chosen action.

Fact Instance: req

Create a choice.

14. Click OK.
15. Set the global variable to an appropriate value. For an overview of how to set global variables, see Step 7 through Step 9 on page 18-29.

Task 4: Create the RL Function Contract

You now create an RL function.

1. Select Definitions.
2. Select RLFunction in the Definitions tree.
   The RLFunction Summary window appears.
3. Click Create.
4. Enter the following details:
Methodology for Rule Set Modeling and Integration with a BPEL Process

5. Add the following function to the Function Body section:

// Create JAXB object factory and result object
creditrating.ObjectFactory of = new creditrating.ObjectFactory();
creditrating.Rating result = of.createRating();
// Get current calendar
java.util.Calendar calendar = java.util.Calendar.getInstance();
// Create new RL object and convert from JAXB to RL
MyRatingRequest mrr = new MyRatingRequest();
mrr.ssn = request.getSSN();
mrr.name = request.getName();
mrr.age = request.getAge();
mrr.amount = request.getAmount();
mrr.date = request.getDate().getTimeInMillis();
// Assert the RL object and run Alternate Ruleset
assert(mrr);
run("AlternateRuleset");
// Result is in variable theResult, convert back to JAXB and return
result.setRating(theResult.rating);
result.setRisk(theResult.risk);
result.setMaxAmount(theResult.maxAmount);
result.setDate(calendar);
return result;

6. Click OK.

This function can be used from Oracle BPEL Process Manager since it has an XML contract, although the underlying rule data model is based on RL fact types.

See Also: “Invoking the Sample Rule Set from a BPEL Process” on page 18-33 to integrate the rule set with a BPEL process
Invoking the Sample Rule Set from a BPEL Process

This section describes how to integrate the rule sets created with the Oracle Business Rules Rule Author with a BPEL process.

This section contains the following topics:

- Task 1: Create a Connection to the Rule Engine
- Task 2: Create a BPEL Project
- Task 3: Create a Decision Service Partner Link
- Task 4: Create a Decide Activity

Task 1: Create a Connection to the Rule Engine

1. Restart Oracle JDeveloper.
2. Select Connection Navigator from the View main menu in Oracle JDeveloper.
4. Click Next on the Welcome window.
5. Enter SampleRuleRepository in the Connection Name field.
6. Select Oracle Rules Engine File Repository as the business rule engine to which to connect.
7. Click Next.
   The Connection window appears.
8. Click the folder icon to select the directory in which the file repository is located.
9. Select the CreditRatingRepository file repository from the C:\CreditRatingRules\model\CreditRatingRepository directory. This is the repository you created in Step 1 on page 18-19.
10. Click Open.
11. Click Next.
   The Test Connection window appears.
12. Click Test.
   If the connection to the business rule engine is successful, the following message appears:
   Success
13. Click Finish.

See Also: “Create Rule Engine Connection Wizard” on page 18-7

Task 2: Create a BPEL Project

1. Right-click your application in the Application Navigator section.
2. Select New Project to define a new BPEL process project.
3. Double-click BPEL Process Project in the Items window to display the BPEL Project Creation Wizard window.
4. Enter SampleProcess in the Name field.
5. Select Synchronous BPEL Process from the Template list.
6. Click Next.
7. Click the flashlight icon to the right of the Input Schema Element field.
8. In the Select Schema window, select CreditRatingTypes.xsd from the directory in which you saved it in Step 2 of "Task 1: Create a Data Model for Rule Authoring" on page 18-19 and click Open.
   The Type Chooser window appears.
9. Expand and select Imported Schemas > CreditRatingTypes.xsd > ratingrequest.
10. Click OK.
11. Click the flashlight icon to the right of the Output Schema Element field.
    The Type Chooser window appears.
12. Expand and select Imported Schemas > CreditRatingTypes.xsd > rating.
13. Click OK.
14. Click Finish.
   This completes the BPEL project creation wizard. A new BPEL process template is created for you with a receive activity accepting a ratingrequest element and a reply activity sending out a rating element.

Task 3: Create a Decision Service Partner Link
1. Ensure that Services is selected in the drop-down list of the Component Palette section in the upper right section of Oracle JDeveloper.
2. Drag and drop a Decision Service onto the right side of the designer window anywhere beneath the header Services.
   The Select a Ruleset or a Function window appears. This window enables you to select an invocation pattern.
3. Enter the following details:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service Name</td>
<td>CreditRatingService</td>
</tr>
<tr>
<td>Namespace</td>
<td><a href="http://xmlns.oracle.com/myLoanProcess/CreditRatingService">http://xmlns.oracle.com/myLoanProcess/CreditRatingService</a></td>
</tr>
<tr>
<td>Invocation Pattern</td>
<td>Execute Ruleset</td>
</tr>
</tbody>
</table>

4. Click the flashlight icon next to the Ruleset field.
   The Rule Explorer window appears.
   This window enables you to browse and select the rule set in the dictionary of the repository you modified in Step 3 of "Task 4: Create a New Rule Set and Model Rules" on page 18-23.
5. Click Show All Versions at the bottom of the window to display all versions of rule dictionaries in the specified repository in the business rule engine. Business rule engines can contain multiple rule dictionaries and versions.

7. Click OK.

   You are returned to the Select a Ruleset or a Function window of the Decision Service wizard. All fact names for the `SampleRuleset` rule set now appear in the fact table.

8. Specify the details of the interaction pattern.
   - Select the Assert Fact check box for the `RatingRequest` fact type.
     This asserts a fact to the rule set (sends an input parameter of factual data to the business rule engine).
   - Select the Watch Fact check box for the `Rating` fact type.
     This returns results created by the business rule engine as part of executing the rule set `SampleRuleset`.

9. Click Next.

   The Copy XSD Files window shows the directory path to the `CreditRatingTypes.xsd` schema file for the wizard to import. If the wizard cannot find this file, you must manually copy it to the `rules/xsd` directory of the `SampleProcess` BPEL project.

10. Click Next, then Finish.
A partner link of the name you specified in Step 3 on page 18-34 is created. This partner link provides the interface between the BPEL process and the business rule engine.

11. Select **Save** from the File main menu.

See Also:
- "Decision Service Wizard" on page 18-9
- "Decision Service Partner Link Directory Structure" on page 18-39

**Task 4: Create a Decide Activity**

1. Drag and drop a **Decide** activity from the **Process Activities** list of the **Component Palette** to below the **receiveInput** receive activity in the designer window.

   The Edit Decide window appears.

2. Enter the following values:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>GetRating</td>
</tr>
<tr>
<td>Decision Service</td>
<td>CreditRatingService</td>
</tr>
<tr>
<td>Operation</td>
<td>Assert facts, execute rule set, retrieve results, and reset the session</td>
</tr>
</tbody>
</table>

3. Click **Assign Input Facts**.

   You now map BPEL input variables to automatically created BPEL variables that correspond to the **Ratingrequest** input (assert) fact type.

4. Click **Create**.

   The **From** section shows the BPEL variables of the process and the **To** section shows the facts selected for the partner link interaction. Since you reused the XML schema file of the fact types in your BPEL process, you can assign the top level **ratingrequest** element from the BPEL input variable **inputVariable** to the fact to assert.

5. Enter the following details:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>From</td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>Variable</td>
</tr>
</tbody>
</table>
The window appears as follows:

6. Click **OK**.

7. Click **Assign Output Facts**.

You now assign the results of executing the rule set to a BPEL variable. For the assignment of output facts, the **From** section displays the facts as modeled in the decision service partner link interaction and the **To** section lists the variables of the BPEL process.

8. Enter the following details:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>From</td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>Variable</td>
</tr>
<tr>
<td>Variables</td>
<td>Expand and select Variables &gt; creditrating_Ratingrequest_Ratingrequest_i &gt; ns1:creditrating_Ratingrequest_i</td>
</tr>
</tbody>
</table>

**Note:** The namespace number values (for example, ns1, ns2) can vary. Use the namespace values that automatically appear.

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>To</td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>Variable</td>
</tr>
<tr>
<td>Variables</td>
<td>Expand and select Variables &gt; creditrating_Ratingrequest_Ratingrequest_i &gt; ns1:creditrating_Ratingrequest_i</td>
</tr>
</tbody>
</table>

**Field Value**

- **From**
  - Type: Variable
  - Variables: Expand and select Variables > inputVariable > payload > ns2:ratingrequest

**Note:** The namespace number values (for example, ns1, ns2) can vary. Use the namespace values that automatically appear.

- **To**
  - Type: Variable
  - Variables: Expand and select Variables > creditrating_Ratingrequest_Ratingrequest_i > ns2:ratingrequest
9. Click OK to close the Decision Fact Map window and Edit Decide window. This completes decide activity configuration. The BPEL process model is updated with the `GetRating` decide activity and the BPEL process is ready for deployment.
Summary of Methodology

This section provides a summary of issues to consider before designing a rule set.

- Plan the data model before starting to model business rules.
- Specify the data model using XML schema constructs.
- Import the XML schema into the rule author as XML fact types.
- Keep limitations of JAXB in mind (element and complex type naming, target package for multiple schemata, and so on).
- Specify RL functions that can be used in rule patterns and actions.
- Model a rule that does not include an if statement test pattern and set it to low priority to generate a default result in the case where no other rule starts as part of rule set execution.
- Model your rules on the XML schema type level.
- Be aware that the data exchange with the BPEL world is on the XML element level.
- Be aware that an alternative approach using RL functions can be taken in the case of an existing data model based on RL or Java fact types.

Decision Service Deployment and Run Time

This section describes decision service partner link creation, deployment, and run time issues.

This section contains the following topics:

- Decision Service Partner Link Directory Structure
- Deployment
- Run Time

Decision Service Partner Link Directory Structure

As part of decision service partner link creation during design time, subdirectories and files are created under the following directory:

\%Dev_Oracle_Home\%mywork\%application_name\%
\%process_name\%decisionservices\%DecisionService

- ear — Top level directory for the J2EE enterprise archive (EAR)
- war — Top level directory for the Web archive (WAR)

where DecisionService is the name you entered in the Service Name field of the Decision Service wizard.

Table 18-1 shows the subdirectories and files of the ear and war directories.

See Also: "Decide Activity" on page 18-14
The following steps are automatically performed as part of decision service partner link creation:

- A new directory structure (see Table 18–1) is created in the following directory:
  \$Dev_Oracle_Home/jdev/mywork/application_name/process_name/decisionservices

- EAR deployment descriptors are generated and stored in the META-INF subdirectory of the enterprise archive.

- A Java server page file GetDecisionServiceInfo.jsp is generated and stored in the public_html directory of the Web archive.

- The decision service-dependent WSDL file DecisionService.wsdl is generated and stored in the WEB-INF/wsdl directory of the Web archive. All dependent XML schema files are also copied to that directory. Dependent schema files include the definitions for the Web service messages and contract and the XML schema files for the XML fact types of the business rule engine.

- The decision service configuration file decisionservices.xml is generated and copied to the directory WEB-INF/classes of the Web archive.

- The rule repository location is resolved:
  - If the rule repository is a file repository, the repository file is copied from its original location to the WEB-INF/repository directory of the Web archive and the configuration file decisionservices.xml is modified to reference the new location.
  - If the rule repository is a WebDav repository, the configuration file is not edited.

- JAXB generation steps:
  - A list of XML schema files for the XML fact types being used in the partner link is obtained.
Oracle JAXB generator is used to generate JAXB classes for the XML fact types in the directory WEB-INF/classes of the Web archive.

- Web service deployment descriptors and the JAX-RPC mapping file are generated in the directory WEB-INF of the Web archive.

The decisionservices.xml decision service configuration file includes the necessary information to generate an interaction pattern-specific WSDL.

From the following information in the configuration file:

```xml
<pattern name="AssertExecuteWatchStateless">
  <arguments>
    <assert>creditrating.Ratingrequest</assert>
    <watch>creditrating.Rating</watch>
  </arguments>
</pattern>
```

you can understand the messages and operations for WSDL generation.

In the following section, the XML schema file of the XML fact types is imported into the decisionservices.wsdl file. This enables the fact elements to be referenced:

```xml
<types>
  <schema xmlns="http://www.w3.org/2001/XMLSchema" ...>
    <include schemaLocation="CreditRatingAgentTypes.xsd"/>
  </schema>
  <schema xmlns="http://www.w3.org/2001/XMLSchema" ...>
    <import namespace="http://samples.otn.com/bpel/demo"
      schemaLocation="CreditRatingTypes.xsd"/>
    <import namespace="http://xmlns.oracle.com/bpel"/>
  </schema>
<types>
```

In this section, the XML schema element corresponding to the fact `creditrating.Ratingrequest` is shown:

```xml
<element name="assertExecuteWatchStateless">
  <complexType>
    <sequence>
      <element name="configURL" type="string" maxOccurs="1"/>
      <element name="bpelInstance" type="bpelpm:tBpelProcess" maxOccurs="1"/>
      <element name="assertList" minOccurs="1" maxOccurs="1">
        <complexType>
          <sequence>
            <element ref="ns1:ratingrequest"/>
          </sequence>
        </complexType>
      </element>
    </sequence>
    <attribute name="name" type="NCName" use="required"/>
  </complexType>
</element>
```

In this section, the XML schema element corresponding to the fact `creditrating.Rating` is shown:

```xml
... ...
```
In this section, appropriate messages are created for the interaction pattern:

```xml
<message name="assertExecuteWatchStatelessMessage">
  <part name="payload" element="tns:assertExecuteWatchStateless"/>
</message>
```

```xml
<message name="assertExecuteWatchStatelessDecisionMessage">
  <part name="payload" element="tns:assertExecuteWatchStatelessDecision"/>
</message>
```

```xml
<message name="decisionServiceError">
  <part name="payload" element="tns:errorInfo"/>
</message>
```

In this section, an operation is created for the interaction pattern with input and output messages corresponding to the selected fact types:

```xml
<portType name="IDecisionService">
  <operation name="assertExecuteWatchStateless">
    <input name="assertExecuteWatchStatelessInput" message="tns:assertExecuteWatchStatelessMessage"/>
    <output name="assertExecuteWatchStatelessOutput" message="tns:assertExecuteWatchStatelessDecisionMessage"/>
    <fault name="operationErroredFault" message="tns:decisionServiceError"/>
  </operation>
</portType>
```

Deployment

The decision services modeled in a BPEL project are deployed with the BPEL process. As part of BPEL process deployment, the following steps are performed for all decision services in the project:

- The Java compiler (javac) is used and all Java classes are compiled in the subdirectory WEB-INF/classes.
A Web archive `DecisionService.war` file is created in the `ear` enterprise archive subdirectory of a decision service. The Web archive consists of all files under the `war` directory of a decision service.

An enterprise archive `DecisionService.ear` file is created in the top level directory of a decision service. The enterprise archive consists of all files in the `ear` directory of a decision service, plus the Web archive created above.

The enterprise archive `DecisionService.ear` is deployed to the underlying J2EE container using the administrator tools of the specific container.

The J2EE context root of a decision service `DecisionService` is as follows:

```
http://${hostname}:${http_port}/rules/${domain_id}/${process_id}/${process_revision}/DecisionService
```

The parameters for this syntax are defined as follows:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>${hostname}</td>
<td>The name of the host that on which the application server is installed</td>
</tr>
<tr>
<td>${http_port}</td>
<td>The HTTP port of the application server</td>
</tr>
<tr>
<td>${domain_id}</td>
<td>The BPEL domain</td>
</tr>
<tr>
<td>${process_id}</td>
<td>The BPEL process name</td>
</tr>
<tr>
<td>${process_revision}</td>
<td>The BPEL process version</td>
</tr>
</tbody>
</table>

The decision services for a specific BPEL process and revision can be identified using Oracle Enterprise Manager 10g Grid Control Console.

See Also: “Run Time” on page 18-43

Run Time

The decision service run time component is a standard J2EE JAX-RPC Web service. This section describes how to manage the decision service from the following consoles:

- Oracle Enterprise Manager 10g Application Server Control Console Support
- Oracle BPEL Control Support

See Also: The AutoLoanDemo tutorial, which describes how to design a BPEL process that integrates with business rules and uses human workflow:

```
SQA_Oraclt_Home\bpel\samples\demos\AutoLoanDemo
```

Oracle Enterprise Manager 10g Application Server Control Console Support

There are several implications of deploying decision services as self-contained enterprise archives. The most important is that every decision service can be managed separately and independently of its invoking BPEL process using Oracle Enterprise Manager 10g Application Server Control Console.

For example, with the AutoLoanDemo sample included as part of Oracle BPEL Process Manager, the AutoLoanFlow BPEL process consists of two decision service partner links.
Decision Service Deployment and Run Time

- Loan advisor
- Credit rating

1. Log into Oracle Enterprise Manager 10g Application Server Control Console.
   
   http://hostname:<port>/em

2. Click Web Services.
   
   The two decision services appear: LoanAdvisorAgent and CreditRatingAgent.

3. Click CreditRatingAgentPort to receive more details for the CreditRatingAgent decision service:

   You can manage and diagnose the decision services as with any other Web service.

4. Click Operations to access additional details.
Oracle BPEL Control Support
You can monitor and diagnose decision services through Oracle BPEL Control. A decisionServiceDetails property is added to the BPEL suitcase that refers to the configuration information of a decision service partner link.

1. Log into Oracle BPEL Control.
2. Click the AutoLoanFlow BPEL process in Oracle BPEL Control.
3. Click Descriptor.
   
   The Descriptor tab shows the process descriptor of the AutoLoanFlow process included with the AutoLoanDemo sample:

   The process has two decision service partner links configured:
   - CreditRatingAgentPL for credit rating
   - LoanAdvisorAgentPL for loan advisory

4. Access additional details about a decision service partner link by clicking Rule Service Details (for example, the details for CreditRatingAgentPL).
### Decision Service Deployment and Run Time

<table>
<thead>
<tr>
<th>CreditAdvisorAgent</th>
<th>Oracle BPEL Process Manager Developer's Guide</th>
</tr>
</thead>
</table>

- **Rule Engine Information**
  - Provider: Oracle BPEL Process Manager
  - Repository: `Oracle_BPEL_PROCESS` application
  - Catalog: `RatingsRules`
  - Version: `v1.0`
  - Subset: `Approved_RatingRules`

- **Interaction Information**
  - Input: `creditRatingRatingRequest`
  - Output: `creditRatingRating`

The information displayed includes the following:

- **Rule engine information**
  - The backend rule engine provider (Oracle in this case)
  - The physical location of the rule repository
  - The name of the rule catalog being used for that partner link
  - (Optional) The version of the rule catalog being used
  - (Optional) The rule set used by the partner link

- **Interaction information**
  - The interaction patterns used by the partner link
  - The input and output fact types used per interaction pattern

5. If you want to open the rule author and update the rule set, click **Open Rule Author**.

   Decide activity details are also available in the Flow window.

6. Click the **Instances** tab.

7. Click a specific instance in the **Instance** list.

8. Click **Flow**.

9. Click a decide activity in the process instance flow to access the same information (for this example, named **LoanAdvisorAgent**).
Advanced Decision Service Features

This section describes advanced decision service features for which limited or no user interface support is provided. Instead, you manually edit deployment description and configuration files to use these features.

This section contains the following topics:

- Using WSIF Bindings
- Enabling Logging of Oracle Business Rules Rule Session Events
- Customizing assertXPath

Using WSIF Bindings

As described in "Decision Service Components" on page 18-4, decision services are JAX-RPC Web services. Therefore, SOAP is the protocol to use with a decision service. However, you can configure the BPEL process to use the decision service in a WSIF context.

Perform the following procedures:
Advanced Decision Service Features

1. Remove the wsdlRuntimeLocation property for a decision service partner link from the bpel.xml deployment descriptor file of the BPEL process.

2. Add fact type Java classes to the classpath of Oracle BPEL Process Manager.
   a. If the decision service partner link was deployed before, you can copy all the files from:

   JDev_Oracle_Home\jdev\mywork\application_name\process_name\decisionservices\DecisionService\war\WEB-INF\classes

   to

   SOA_Oracle_Home\bpel\system\classes

   b. Otherwise, you must compile the Java classes located in the following directory:

   JDev_Oracle_Home\jdev\mywork\application_name\process_name\decisionservices\DecisionService\war\WEB-INF\classes

   to

   SOA_Oracle_Home\bpel\system\classes

3. Deploy the BPEL process.

Enabling Logging of Oracle Business Rules Rule Session Events

The Oracle Business Rules Rules Engine defines several rule session events for monitoring. (See Oracle Business Rules Language Reference Guide for additional details.) The decision service provides the option to enable these events and log the output to the Oracle BPEL Process Manager log file.

The events are enabled through properties in the decision service configuration file (decisionservices.xml). Table 18–2 describes the properties that can be set.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>watchRules</td>
<td>Information about rule invocations (execution of activations)</td>
</tr>
<tr>
<td>watchActivations</td>
<td>Addition or removal of activations from the agenda</td>
</tr>
<tr>
<td>watchFacts</td>
<td>Assertion, retraction, or modification of facts in working memory</td>
</tr>
<tr>
<td>watchFocus</td>
<td>Pushing or popping of the rule set stack</td>
</tr>
<tr>
<td>watchCompilations</td>
<td>When a rule’s conditions are added to the network, information about how the condition parts are shared with existing rules is printed. “=” indicates sharing.</td>
</tr>
<tr>
<td>watchAll</td>
<td>Includes information from all of the above events</td>
</tr>
</tbody>
</table>

1. Open the file:

   JDev_Oracle_Home\jdev\mywork\application_name\process_name\decisionservices\DecisionService\war\WEB-INF\classes\decisionservices.xml

2. Add the properties after the repository element in the ruleEngineProvider section.

   <?xml version = '1.0' encoding = 'UTF-8'?>
<decisionServices xmlns="http://xmlns.oracle.com/bpel/rules">
  <ruleEngineProvider name="CreditRatingRuleRepository"
    provider="Oracle">
    <repository type="File">
      <file>repositoryresource:CreditRatingRepository</file>
    </repository>
  </ruleEngineProvider>
  <decisionService name="CreditRatingAgent"
    targetNamespace="http://xmlns.oracle.com/AutoLoanFlow/CreditRatingAgent"
    ruleEngineProviderReference="CreditRatingRuleRepository">
    <catalog>RatingFY06</catalog>
    <catalogVersion>Approved_060205</catalogVersion>
    <ruleset>PrivateCustomerRatingRules</ruleset>
    <pattern name="AssertExecuteWatchStateless">
      <arguments>
        <assert>creditrating.Ratingrequest</assert>
        <watch>creditrating.Rating</watch>
      </arguments>
    </pattern>
  </decisionService>
</decisionServices>

3. Redeploy the decision service. The following is a sample output taken from the
credit rating agent of the AutoLoanFlow process.

```
<2006-07-05 10:18:13,710> <DEBUG> <default.collaxa.cube.services>
  <OracleRuleSession::executeUnitOfWork>
    Execution plan for AutoLoanFlow:202
  </OracleRuleSession::executeUnitOfWork>
</default.collaxa.cube.services>
<2006-07-05 10:18:13,710> <DEBUG> <default.collaxa.cube.services>
  assert fact creditrating.Ratingrequest
</default.collaxa.cube.services>
<2006-07-05 10:18:13,750> <DEBUG> <default.collaxa.cube.services>
  ==> Activation: creditrating.xpath.retractDeadRatingrequest : f-1
</default.collaxa.cube.services>
<2006-07-05 10:18:13,750> <DEBUG> <default.collaxa.cube.services>
  ==> Activation: creditrating.xpath.retractDeadRatingrequestType : f-1
</default.collaxa.cube.services>
<2006-07-05 10:18:13,790> <DEBUG> <default.collaxa.cube.services>
  ==> Activation: PrivateCustomerRatingRules.Default : f-1
</default.collaxa.cube.services>
<2006-07-05 10:18:13,790> <DEBUG> <default.collaxa.cube.services>
  ==> Activation: PrivateCustomerRatingRules.YoungCustomers : f-1
</default.collaxa.cube.services>
<2006-07-05 10:18:13,790> <DEBUG> <default.collaxa.cube.services>
  ==> Activation: PrivateCustomerRatingRules.HighRiskCustomers : f-1
</default.collaxa.cube.services>
<2006-07-05 10:18:13,790> <DEBUG> <default.collaxa.cube.services>
  Execute rule set PrivateCustomerRatingRules
</default.collaxa.cube.services>
```

BPEL Process Integration with Business Rules  18-49
Oracle Business Rules can specify an XPath expression when asserting facts. This reduces the number of assertions and provides a convenient mechanism to assert multiple facts with a single assert statement.

This functionality is available in Oracle JDeveloper. Select Check here to assert all descendants from the top level element on the Select a Ruleset or a Function window in the Decision Service wizard. When you select this option, a default XPath "//*" is created for the fact to assert. This causes all descendants of the fact element to assert during run time.

You can customize the XPath expression manually by modifying the decision service configuration located in the following file:

```
JDev_Oracle_Home/jdev/mywork/application_name/process_name/decisionservices/DecisionService/web-INF/classes/decisionservices.xml
```

The following example shows this option enabled for a fact type.

```xml
<repository type="file">
  <file>repositoryresource:CreditRatingRepository</file>
</repository>
```

```
<ruleEngineProvider name="CreditRatingRuleRepository" provider="Oracle">
  <repository type="File">
    <file>repositoryresource:CreditRatingRepository</file>
  </repository>
</ruleEngineProvider>
```
Example of BPEL Process Integration with Business Rules

You can customize the attribute xpath="/**" before deploying the decision service.

Example of BPEL Process Integration with Business Rules

The section describes how to design and integrate a BPEL process with the business rules of a business rule engine. This example is part of a larger tutorial that also describes how to design this BPEL process to use human workflow. Only the part describing BPEL process integration with business rules is included in this section.

This section contains the following topics:

- Task 1: Update a Rule Using Oracle Business Rules Rule Author
- Task 2: Create a Connection to the Business Rule Repository
- Task 3: Create a BPEL Process and Import the Schema
- Task 4: Create a Decision Service Partner Link
- Task 5: Create a Decide Activity

See Also: The complete AutoLoanDemo tutorial, which describes how to design a BPEL process that integrates with business rules and uses human workflow:

SOA_Oracle_Home/bpel/samples/demos/AutoLoanDemo

Task 1: Update a Rule Using Oracle Business Rules Rule Author

This section describes how to access the Oracle Business Rules Rule Author and modify a business rule that you later integrate with your BPEL process.

1. Log into the Oracle Business Rules Rule Author.

   http://hostname:port/ruleauthor

   The Oracle Business Rules Rule Author is automatically installed with the SOA Suite Basic Install type.

2. Log in as oc4jadmin/password.

   where password is the oc4jadmin password you entered during installation.

3. Click the Repository tab at the top.

4. Select File from the Repository Type list.
5. Click **Browse** and select the **CreditRatingRepository** file repository from the

```
SOA_Oracle_Home\bpel\samples\demos\AutoLoanDemo\repository
```

directory.

This page displays the following details:

**Connect**

Please connect to a repository to load or create a dictionary before performing any other operations. You have to reload the dictionary if your session expires due to inactivity.

![Repository Type: File, File Location: SOA_Oracle_Home\bpel\samples\demos\AutoLoanDemo\repository](image)

6. Click **Connect**.

A message displays indicating that a connection has been made to the repository.

7. Click the **Customization** tab at the top.

8. Enter the following details to load the rule dictionary and its versions from the

repository selected in Step 5:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Existing Dictionaries</td>
<td>RatingFY06</td>
</tr>
<tr>
<td>Version</td>
<td>Approved.060205</td>
</tr>
</tbody>
</table>

This page displays the following details:

**Load Dictionary**

No dictionary has been loaded in your session. Load a version of dictionary from the repository.

![Existing Dictionaries: RatingFY06, Version: Approved.060205, Description: Rules approved on 02/05/2006](image)

9. Click **Load**.

A confirmation message appears:

```
Dictionary 'RatingFY06 (Approved.060205)' has been loaded
```

10. Click the **Customization** tab at the top. You now modify a rule in the rule set.

11. Click the **YoungCustomers** rule and change `request.customer age <` from `40` to `45`.

This page displays the following details:
12. **Click Apply.**
   A message displays indicating that the customization has been applied.

13. **Click the HighRiskCustomers rule and change request.customer age >= from 40 to 45.**

14. **Click Apply.**

15. **Click Save Dictionary at the top.**

16. **Click Save.**
   This saves the current dictionary contents (including your updates) in the CreditRatingRepository repository.

17. **Click Logout.**

**Task 2: Create a Connection to the Business Rule Repository**

This section describes how to create a connection to the business rule repository.

1. Select **Connection Navigator** from the View main menu in Oracle JDeveloper.

2. **Right-click Rule Engines and select New Rule Engine Connection.**

3. **Click Next on the Welcome window.**

4. **Enter CreditRatingRuleRepository in the Connection Name field.**

5. **Select Oracle Rules Engine File Repository as the type of business rule engine to which to connect.**
6. Click Next.
   The Connection window appears.

7. Click the folder icon to select the directory in which the file repository is located.

8. Select the CreditRatingRepository file repository from the SOA_Oracle_Home\bpel\samples\demos\AutoLoanDemo\repository directory. This is the repository you loaded in Step 5 on page 18-52.

9. Click Open.

10. Click Next.
    The Test Connection window appears.

11. Click Test.
    If the connection to the business rule repository is successful, the following message appears:

        Success
Task 3: Create a BPEL Process and Import the Schema

You now create a BPEL process in which to integrate the business rules of the business rule engine you modified in "Task 1: Update a Rule Using Oracle Business Rules Rule Author" on page 18-51.

1. Right-click your application in the Application Navigator section.
2. Select New Project to define a new BPEL process project.
3. Double-click BPEL Process Project in the Items window to display the BPEL Project Creation Wizard window.
4. Enter AutoLoanFlow in the Name field. All other fields default to the correct values for creating an asynchronous BPEL process.
5. Click Next.
6. Click the flashlight icon to the right of the Input Schema Element field.
7. In the Select Schema window, select AutoLoanTypes.xsd from the SOA_Oracle_Home\bpel\samples\demos\AutoLoanDemo\AutoLoanFlow\bpel directory and click Open.
   The Type Chooser window appears.
8. Expand and select Imported Schemas > AutoLoanTypes.xsd > loanApplication.
9. Click OK.
10. Click the flashlight icon to the right of the Output Schema Element field.
    The Type Chooser window appears.
11. Expand and select Imported Schemas > AutoLoanTypes.xsd > loanOffer.
12. Click OK.
13. Click Finish.

Task 4: Create a Decision Service Partner Link

You now use the Decision Service Wizard to connect to the business rule engine and convert the rule set you modified in "Task 1: Update a Rule Using Oracle Business Rules Rule Author" into a Web service to use in the BPEL process. When complete, a decision service partner link is created.

1. Ensure that Services is selected in the drop-down list of the Component Palette section in the upper right section of Oracle JDeveloper.
2. Drag and drop a Decision Service onto the right side of the designer window anywhere beneath the header Services.
   The Select a Ruleset or a Function window appears.
   This window enables you to select an invocation pattern.
3. Enter the following details:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service Name</td>
<td>CreditRatingAgent</td>
</tr>
<tr>
<td>Note</td>
<td>When complete, this becomes the name of the partner link.</td>
</tr>
</tbody>
</table>
Example of BPEL Process Integration with Business Rules

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Namespace</td>
<td><a href="http://xmlns.oracle.com/myLoanProcess/CreditRatingAgent">http://xmlns.oracle.com/myLoanProcess/CreditRatingAgent</a></td>
</tr>
<tr>
<td>Note</td>
<td>This field is automatically completed with your entry in the</td>
</tr>
<tr>
<td></td>
<td>Service Name field.</td>
</tr>
<tr>
<td>Invocation Pattern</td>
<td>Execute Ruleset</td>
</tr>
</tbody>
</table>

The window now appears as follows:

4. Click the flashlight icon next to the Ruleset field.
   The Rule Explorer window appears.
   This window enables you to browse and select the rule set in the dictionary of the repository you modified in "Task 1: Update a Rule Using Oracle Business Rules Rule Author" on page 18-51.

5. Click Show All Versions at the bottom of the window to display all catalog versions of rule dictionaries in the specified repository. Business rule repositories can contain multiple rule dictionaries and versions.

7. Click OK.
You are returned to the Select a Ruleset or a Function window of the Decision Service wizard. Note that all fact names for the PrivateCustomerRatingRules rule set now appear in the fact table.

8. Select the input (Assert Fact) and output (Watch Fact) fact types:
   - Select the Assert Fact check box for the Ratingrequest fact type.
     This asserts a fact to the rule set (sends factual data to the business rule engine).
   - Select the Watch Fact check box for the Rating fact type.
     This returns results from the rule set. This table column only appears because you selected an invocation pattern that retrieves results in Step 3 on page 18-55.
Example of BPEL Process Integration with Business Rules

9. Click Next.
   This window shows the schema file for the wizard to import.

10. Click Next, then Finish.
    A partner link of the name you specified in Step 3 on page 18-55 is created. This partner link provides the interface between the BPEL process and the business rule engine.
11. Select Save from the File main menu.

Task 5: Create a Decide Activity

You now create a decide activity to invoke the decision service partner link you created with the Decision Service wizard. The decide activity enables you to create copy operation assignments between the fact types in your rule set (now included in the partner link) and BPEL variables. You provide an input fact to the rule set and then retrieve the results. This enables you to invoke rules from the BPEL process.

When complete, a decide activity consisting of assign and invoke activities to the decision service partner link is created.

1. Drag and drop a Decide activity below the receiveInput receive activity in the designer window.
2. Enter the following values:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>GetCreditRating</td>
</tr>
<tr>
<td>Decision</td>
<td>CreditRatingAgent</td>
</tr>
<tr>
<td>Service</td>
<td>Assert facts, execute rule set, retrieve results, and reset the session</td>
</tr>
</tbody>
</table>

3. Click Assign Input Facts.
   You now map BPEL input variables to automatically created BPEL variables that correspond to the Ratingrequest input (assert) fact type.
4. Click Create.
5. Enter the following details:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>From Type</td>
<td>Variable</td>
</tr>
<tr>
<td>Variables</td>
<td>Expand and select Variables &gt; inputVariable &gt; payload &gt; ns1:LoanApplication &gt; ns1:SSN</td>
</tr>
<tr>
<td>Note: The namespace number values (for example, ns1, ns2) can vary. Use the namespace values that automatically appear.</td>
<td></td>
</tr>
<tr>
<td>To Type</td>
<td>Variable</td>
</tr>
<tr>
<td>Variables</td>
<td>Expand and select Variables &gt; ratingrequest_i &gt; ns3:ratingrequest &gt; ns3:SSN</td>
</tr>
</tbody>
</table>

The window appears as follows:
6. Click OK.
7. Click Create again to create a second copy operation.
8. Enter the following details:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>From</td>
<td></td>
</tr>
<tr>
<td>• Type</td>
<td>Variable</td>
</tr>
<tr>
<td>• Variables</td>
<td>Expand and select Variables &gt; inputVariable &gt; payload &gt; ns1:loanApplication &gt; ns1:customerName</td>
</tr>
<tr>
<td>To</td>
<td></td>
</tr>
<tr>
<td>• Type</td>
<td>Variable</td>
</tr>
<tr>
<td>• Variables</td>
<td>Expand and select Variables &gt; creditrating_Ratingrequest_i &gt; ns3:ratingrequest &gt; ns3:name</td>
</tr>
</tbody>
</table>

9. Click OK
10. Click Create again to create a third copy operation.
11. Enter the following details:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>From</td>
<td></td>
</tr>
<tr>
<td>• Type</td>
<td>Variable</td>
</tr>
</tbody>
</table>

12. Click Apply.

The Edit Decide window displays the following input fact mappings:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variables</td>
<td>Expand and select Variables &gt; inputVariable &gt; payload &gt; ns1:loanApplication &gt; ns1:loanAmount</td>
</tr>
<tr>
<td>To</td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>Variable</td>
</tr>
<tr>
<td>Variables</td>
<td>Expand and select Variables &gt; critrating_Ratingrequest_i &gt; ns3:ratingrequest &gt; ns3:amount</td>
</tr>
</tbody>
</table>

13. Click Assign Output Facts.

You now map the automatically created BPEL variables that correspond to the Rating output fact type to BPEL input variables.

14. Click Create.

15. Enter the following details to create the output facts:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>From</td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>Variable</td>
</tr>
<tr>
<td>Variables</td>
<td>Expand and select Variables &gt; critrating_Rating request_o &gt; ns2:rating &gt; ns2:rating</td>
</tr>
</tbody>
</table>

Note: The namespace number values (for example, ns1, ns2) can vary. Use the namespace values that automatically appear.

| To        |       |
| Type      | Variable |
Example of BPEL Process Integration with Business Rules

16. Click OK.
17. Click Create again to create a second copy operation.
18. Enter the following details:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variables</td>
<td>Expand and select Variables &gt; inputVariable &gt; payload &gt; ns3:loanApplication &gt; ns3.creditRating</td>
</tr>
</tbody>
</table>

The window appears as follows:

19. Click OK.
20. Click Create again to create a third copy operation.
21. Enter the following details:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
</table>
| From         | Type Variable
  Variables   | Expand and select Variables > creditrating_Rating_o > ns2:rating > ns2:risk |
| To           | Type Variable
  Variables   | Expand and select Variables > inputVariable > payload > ns3:loanApplication > ns3.creditRisk |

18-62 Oracle BPEL Process Manager Developer's Guide
Example of BPEL Process Integration with Business Rules

22. Click OK.

The Edit Decide window displays the following output fact mappings:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>From</td>
<td></td>
</tr>
<tr>
<td>■ Type</td>
<td>Variable</td>
</tr>
<tr>
<td>■ Variables</td>
<td>Expand and select Variables &gt; creditrating_Rating.o &gt; ns2:rating &gt; ns2:maxAmount</td>
</tr>
<tr>
<td>To</td>
<td></td>
</tr>
<tr>
<td>■ Type</td>
<td>Variable</td>
</tr>
<tr>
<td>■ Variables</td>
<td>Expand and select Variables &gt; inputVariable &gt; payload &gt; ns3:loanApplication &gt; ns3:creditMaxAmount</td>
</tr>
</tbody>
</table>

23. Click OK.

When complete, a decide activity consisting of assign and invoke activities to the decision service partner link is created.
24. Click the + sign to expand the GetCreditRating decide activity and view the assign and invoke activities.
   The BPEL process is now integrated with the business rules of the business rule engine. If you later modify the contents of the business rule, you do not need to redesign your BPEL process.

25. Select Save from the File main menu.
Part IV
Development and Deployment Life Cycle

This part describes how to run and manage BPEL processes from Oracle BPEL Control. This part contains the following chapters:

- Chapter 19, "BPEL Process Deployment and Domain Management"
- Chapter 20, "Testing BPEL Processes"
- Chapter 21, "Oracle BPEL Portlets"
- Chapter 22, "Oracle BPEL Control Reports"
This chapter provides an overview of key BPEL process deployment and domain management concepts. An overview of Oracle BPEL Control from which you can manage processes and domains is also provided. In addition, an overview of several build and command line tools is also provided.

This chapter contains the following topics:

- Compiling and Deploying a BPEL Process
- Creating and Managing a BPEL Domain
- Managing Processes in Oracle BPEL Control
- Build and Command Line Tools
- Summary

**See Also:** The following documentation for tutorials in which you deploy BPEL processes:

- Oracle BPEL Process Manager Order Booking Tutorial
- Oracle BPEL Process Manager Quick Start Guide

### Compiling and Deploying a BPEL Process

After you complete the design of your BPEL process, you compile and deploy the process to Oracle BPEL Server. If compilation and deployment are successful, you can run and manage the BPEL process from Oracle BPEL Control.

Deployment sends the Oracle BPEL Process Manager archive (a set of files in a JAR file with a directory structure similar to the project directory structure) to Oracle BPEL Server. The deployment operation automatically validates and compiles the project directory into the BPEL archive. Therefore, you do not need to explicitly validate, compile, and recompile a project before deployment. Use Oracle BPEL Control to view any currently running BPEL processes before compiling and deploying additional processes.

BPEL processes can be compiled and deployed in Oracle JDeveloper.

**Note:** You must wait for deployment of one BPEL process to complete before attempting to deploy another process. Attempting to deploy a second process while the first process is still deploying can cause problems.
Compiling and Deploying in Oracle JDeveloper

To compile and deploy a BPEL process in Oracle JDeveloper, right-click the BPEL project (for this example, named OrderBooking) in the Application Navigator and select Deploy:

You have two deployment methods from which to choose:

- You can deploy directly to the default domain or any other domain you have created by using an integration server connection.

Domains enable you to partition and manage instances of your processes. A discussion on the importance of domains is provided later in this chapter.

If this is the first time you have deployed this BPEL process to Oracle BPEL Server, a default version label of 1.0 is automatically created. A version identifies a specific deployed instance of a BPEL process. The version label is appended to the end of the JAR file name created when you deploy the BPEL process.

If this label version is already deployed and the server mode is production, you are prompted to either overwrite the existing version or enter a different version label:

If you overwrite the version, the old process definition on the server is replaced by the new definition. You cannot revert to the old definition. In addition, any process instances that ran under the old definition are marked as stale. The stale instances cannot be examined, and all flow and audit information is lost. If you enter a different version label for the new process definition (for example, 2.0), it is deployed to Oracle BPEL Server, while the older, deployed process definition (1.0) also continues to run simultaneously on Oracle BPEL Server. The instances that ran under the old definition are retained, and not marked as stale. You can still examine the flow and audit information for these instances.

If the server mode is development, you are not prompted and the version is automatically overwritten.
This is a key benefit of versioning. For example, you may have an older instance of a BPEL process running with one customer that is still valid. You then begin a partnership with a different customer that requires a slight modification to the design of this BPEL process. At some point you plan to migrate the old customer to the newer version of the BPEL process, but for now that is not necessary. Versioning enables you to run both processes.

If you want to use a more descriptive version name for a process, right-click the process again in the Application Navigator and select Deploy > connection_name > Deploy to domain_name domain. Provide a more descriptive name when prompted in the Your version field of the Deployment Properties window (for example, sales_div_1.0). You can then reire the other process version on Oracle BPEL Control.

- If you select BPEL Process Deployer, the BPEL Process Deployer window opens. This window enables you to customize your settings by selecting a different or creating a new Oracle BPEL Server connection and deploying to domains other than default. If this process version is already deployed, you can also select to overwrite the existing version or enter a different version label to enable both to run simultaneously.

After you select a deployment method, the Log Window at the bottom of Oracle JDeveloper displays messages about the status of the deployment. For example, the following message under the Messages tab indicates that deployment was successful.

```plaintext
```

The following message under the Apache Ant tab also indicates that deployment was successful.

```plaintext
BUILD SUCCESSFUL
Total time: 10 seconds
```

**Caution:** Use caution when reusing version labels in a production environment, due to the potential loss of data. In a development environment, it can be useful to reuse version numbers to avoid creating unnecessary revisions of the process on Oracle BPEL Server.

If deployment is unsuccessful, errors display in the Log Window. Click the error to display the line of code that caused deployment to fail.
Compiling Without Deploying in Oracle JDeveloper

You can also compile without immediately deploying an Oracle BPEL Process Manager archive to Oracle BPEL Server. Perform this action by right-clicking the BPEL process and selecting Make or Rebuild. This places the Oracle BPEL Process Manager Archive in the following directory:

```
JDev_Oracle_Home\jdev\mywork\my_application\project_name\output
```

From this directory, you can deploy the process in either of two ways:

1. Copy the archive to the appropriate domain directory (for this example, default)

   ```
   SOA_Oracle_Home\bpel\domains\default\deploy
   ```

   or

2. Log into Oracle BPEL Control by selecting Start > All Programs > Oracle > Oracle Home > Oracle BPEL Process Manager > BPEL Control.

   2. Click BPEL Processes.
3. Click **Deploy New Process** in the **Related Tasks** section.
4. Click **Browse** to select the BPEL suitcase JAR file for the process, then click **Open**.
5. Click **Deploy**.
6. Click the **Dashboard** tab to view the newly deployed process.

### BPEL Suitcase JAR File

During compilation and deployment, the BPEL process archive and its components are compiled and packaged into a JAR file known as a BPEL suitcase. This JAR file includes the following files:

- `project_name.bpel` file: implementation of the process
- `project_name.wsdl` file
- `bpel.xml` deployment descriptor file
- Any other local resources that are required, such as XML schemas, Java classes or libraries, and so on.

The suitcase JAR file is deployed to the `JDev_Oracle_Home\jdev\mywork\application_name\process_name\output` directory. The suitcase JAR file name follows the convention of `bpel_projectname_versionnumber.jar`. For example:

```
bpel_LoanProcess_1.0.jar
```

**See Also:**

- "Creating and Managing a BPEL Domain" on page 19-8
- "Deploying a BPEL Suitcase to a Specific Domain" on page 19-11

### Deploying to Multiple Environments with Different Configuration Values

When the testing and development phase of a BPEL process has completed, you are ready to deploy the process to a production environment. The configuration properties for the production environment are typically different from those for the development environment. For example, the partner link Web service URL for the production environment is typically different from the one used in the testing environment.

In previous releases, you resolved this conflict by modifying the `bpel.xml` deployment descriptor file or providing separate `bpel.xml` files for the development and production environments. This was a big effort if the `bpel.xml` file required many modifications or if you had multiple versions of `bpel.xml` that required modifications.

Beginning with this release, a **customize ant** task is provided that enables you to specify the property value for development and production environments in a single build file location. The `customize` task captures the changes between different versions of `bpel.xml` using the build file. This task can be used as a subtask of `bpelc` or as a standalone `ant` task. The `bpelc` subtask can contain multiple `customize` tasks. The `customize` task can contain multiple subtasks.

**customize ant Task Example**

This section provides an end-to-end example of using the `customize ant` task.
1. Copy the target domain URL WSDL file of the partner link Web service to the directory in which the bpel.xml file for your BPEL process is located (for this example, IncrementService.wsdl is used).

2. Open the bpel.xml file in the same directory.

3. Modify the directory path for the IncrementService.wsdl file under the partnerLinkBinding element to indicate that this WSDL file is located in the same directory:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<BPELSuitcase>
  <BPELProcess src="Invoke.bpel" id="Invoke">
    <partnerLinkBindings>
      <partnerLinkBinding name="client">
        <property name="wsdlLocation">Invoke.wsdl</property>
      </partnerLinkBinding>
      <partnerLinkBinding name="IncrementService">
        <property name="wsdlLocation">IncrementService.wsdl</property>
      </partnerLinkBinding>
    </partnerLinkBindings>
    <configurations>
      <property name="testIntroduction"><![CDATA[This BPEL process invokes a synchronous integer increment service.]]></property>
    </configurations>
  </BPELProcess>
</BPELSuitcase>
```

4. Create a custom build.xml file in the same directory as the bpel.xml file and the IncrementService.wsdl file that uses the customize task. You specify properties for the host name, port, domain name, and process revision of the partner link Web service that map to values set in the ant-orapbel-properties file:

```xml
<?xml version="1.0" encoding="iso-8859-1"?>
<project name="bpel.deploy" default="compile" basedir="." >
  <target name="compile">
    <bpelc input="${basedir}/bpel.xml" out="${process.dir}/output" rev="${rev}" home="${bpel.home}">
      <customize>
        <partnerLinkBinding name="IncrementService">
          <property name="wsdlLocation">
            http://${host_name}:${port_number}/orabpel/${domain_name}/IncrementService/${rev}/IncrementService?wsdl
          </property>
        </partnerLinkBinding>
      </customize>
    </bpelc>
  </target>
</project>
```

**Note:** You can use your own custom build properties file instead of the ant-orapbel-properties file. Ensure that you import the custom file in the build.xml file.
5. Specify values for the host name, port, domain name, and process revision properties in the SOA_Oracle_Home\bpel\utilities\ant-orapbel-properties file.

```
# Development or production environment
host_name=jsmith-pc.us.oracle.com
port_number=80
domain_name=default
rev=1.0
# END OF FILE
```

This single file is the only location that requires editing between the development and production environments. For example, if the host name for the production environment is different (for example, jwilliams-pc.us.oracle.com), you change it in this file.

6. Go into the main build.xml file located one directory above the bpel.xml, IncrementService.wsdl, and custom build.xml files.

7. Specify a new target name in the process-deploy section (for this example, the default target name of compile is changed to compile1).

```
<target name="process-deploy"
depends="validateTask, compile1, deployProcess, deployTaskForm, deployDecisionServices"
/>  
```

8. Create a new target name section below the target name. For this example, a section with a value of compile1 is specified that calls the custom build.xml script you created in Step 4. The target name section points to the bpel directory that includes the bpel.xml, IncrementService.wsdl, and custom build.xml files.

```
<target name="process-deploy"
depends="validateTask, compile1, deployProcess, deployTaskForm, deployDecisionServices"
/>  
<target name="validateTask">
    <echo>
        | Validating workflow
    </echo>
    <validateTask dir="${process.dir}/bpel"/>
</target>
<target name="compile1">
    <ant dir="${process.dir}/bpel"/>
</target>
```

9. Deploy the BPEL process with ant.

```
ant process-deploy
```

10. Go to Oracle BPEL Control to see that all processes deployed correctly.

**customize ant Task Syntax**

The `customize ant` task can be used to modify several types of properties. This section provides syntax examples of using the `customize ant` task with these properties.
Specifying Custom File Names  By default, the bpel.xml file set in the build.xml file is used. You can also specify custom input and output deployment descriptor files. This action modifies the specified input XML file and writes the results to the specified outFile. Table 19–1 describes the attributes to use.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>inFile</td>
<td>Specifies the input XML file. When used as a subtask of bpelc, the default value is the deployment descriptor file (bpel.xml).</td>
<td>No</td>
</tr>
<tr>
<td>outFile</td>
<td>Specifies the output XML file. When used as a subtask of bpelc, the default value of this is the deployment descriptor file (bpel.xml).</td>
<td>No</td>
</tr>
</tbody>
</table>

Specifying Deployment Descriptor configurations Properties  When used as a bpelc subtask, the customize ant task can add or modify configurations properties of the deployment descriptor.

```xml
<configurations>
  <property name="propName">propValue</property>
</configurations>
```

See Also:  Appendix C, “Deployment Descriptor Properties”

Specifying partnerLinkBinding Properties  When used as a bpelc subtask, the customize ant task can add or modify properties of a partnerLinkBinding.

```xml
<partnerLinkBinding name="partnerLinkName">
  <property name="propName">propValue</property>
</partnerLinkBinding>
```

See Also:  “customize ant Task Example” on page 19-5

Specifying activationAgent Properties  When used as a bpelc subtask, the customize ant task can add or modify properties of an activationAgent.

```xml
<activationAgents name="actAgentName">
  <property name="propName">propValue</property>
</activationAgents>
```

Specifying Deployment Descriptor preferences Properties  When used as a bpelc subtask, the customize ant task can add or modify preferences properties of the deployment descriptor.

```xml
<preferences>
  <property name="propName">propValue</property>
</preferences>
```

See Also:  Appendix C, “Deployment Descriptor Properties”

Creating and Managing a BPEL Domain

BPEL processes (specifically, the suitcase JAR file) are deployed to domains. A BPEL domain allows a developer or administrator to partition a single instance of Oracle BPEL Process Manager into multiple virtual BPEL sections.
Creating and Managing a BPEL Domain

Here are some examples of how to use BPEL domains:

- Partition a single Oracle BPEL Process Manager instance into a multideveloper environment. In this case, the domain ID typically identifies the developer owning that domain.
- Partition a single Oracle BPEL Process Manager instance into both a development and QA environment. In this case, the domain IDs can be test and qa.
- Partition a single Oracle BPEL Process Manager instance into an environment used by multiple departments or partners. In these cases, the domain IDs are the names of the departments or partners.

The following sections describe key BPEL domain issues:

- Logging into Domains
- Changing Domain Passwords
- Creating a BPEL Domain
- Changing Oracle BPEL Server Mode
- Deploying a BPEL Suitcase to a Specific Domain
- Undeploying a BPEL Process from a Specific Domain

Logging into Domains

Oracle BPEL Process Manager domain management and administration is performed from Oracle BPEL Control and Oracle BPEL Admin Console.

Beginning with this release, Oracle BPEL Control and Oracle BPEL Admin Console are secured with Oracle Application Server J2EE and JAAS security features. Access to BPEL domains is now protected through user IDs and passwords. In previous releases, only a password was required.

When Oracle BPEL Process Manager is installed, an initial domain named default is created. The initial password for accessing the default domain through Oracle BPEL Control or any domain you create is the same as that specified for the oc4jadmin user during installation. The procedural instructions described in this chapter for performing tasks use the oc4jadmin user.

You can also use the bpeladmin user or default user and their default password of welcome1 to access domains. The bpeladmin user provides access to all domains and the default user provides access to only the default domain.

The oc4jadmin, bpeladmin, and default users enable you to access Oracle BPEL Control through the following methods:

- Selecting Start > All Programs > Oracle - Oracle_Home > Oracle BPEL Process Manager > BPEL Control
- Going to the following URL:
  http://localhost:port/BPELConsole
  where port is:

  - 8888 if you installed Oracle BPEL Process Manager from the Oracle Application Server SOA software CD.
  - 9700 if you installed the Oracle BPEL Process Manager for Developers or Oracle BPEL Process Manager for OracleAS Middle Tier install type from the Oracle BPEL Process Manager software CD.
Creating and Managing a BPEL Domain

The oc4jadmin and bpeladmin users enable you to access Oracle BPEL Admin Console through the following URL:

http://localhost:port/BPELAdmin

You cannot access Oracle BPEL Admin Console with the default user. Both Oracle BPEL Control and Oracle BPEL Admin Console are described further in subsequent sections of this chapter.

See Also:
- Security chapter of the Oracle BPEL Process Manager Administrator's Guide for additional details about Oracle BPEL Control and Oracle BPEL Admin Console users and the roles that provide access to domains
- Oracle BPEL Process Manager Installation Guide for information about supported Web browsers

Changing Domain Passwords

Passwords for the oc4jadmin, bpeladmin, and default users can be changed through Oracle Enterprise Manager 10g Application Server Control Console. Oracle recommends that you change the passwords for the bpeladmin and default users after installation.

See Also:
- Oracle Application Server Administrator's Guide for instructions on changing the oc4jadmin, bpeladmin, and default passwords
- Oracle BPEL Process Manager Administrator's Guide for additional details about Oracle BPEL Process Manager security

Creating a BPEL Domain

You can create additional domains by performing the following procedures.

1. Access Oracle BPEL Admin Console:
   http://localhost:port/BPELAdmin
2. Enter the oc4jadmin username and password.
3. Click the BPEL Domains tab.
4. Click Create New BPEL Domain.
   The Create New BPEL Domain window appears.
5. Follow the on-screen instructions to create a new domain with an ID.
6. Return to Oracle JDeveloper.
7. Right-click a process.
8. Select Deploy > connection_name > Refresh.
9. Select Deploy > connection_name > Deploy to domain_name domain.
   where domain_name is the ID you entered in Step 5.
10. Log in to Oracle BPEL Control.
11. Select the new domain name from the drop-down list in the upper right corner of Oracle BPEL Control. The process you deployed in Step 9 displays in the Dashboard tab.

**Changing Oracle BPEL Server Mode**

Oracle BPEL Server is automatically installed in production mode. If you attempt to deploy a process in production mode and a label version of that process is already deployed, you are prompted to either overwrite the existing version or enter a different version label.

Follow these instructions to see the current mode of your server in Oracle JDeveloper.

1. Right-click the BPEL process in the Application Navigator.
2. Select Deploy > BPEL Process Deployer.

The Server Mode field of the BPEL Process Deployer window displays the mode.

You can change this mode to development. When you attempt to deploy a process in development mode and a label version of that process is already deployed, it is automatically overwritten and you are not prompted to make a decision.

Follow these instructions to change the current mode of your server.

1. Access Oracle BPEL Admin Console:
   http://localhost:port/BPELAdmin
2. Enter the oc4jadmin username and password.
3. Click the Server tab.
4. Change the productionServer property value to false.
5. Click Apply.
6. Return to the BPEL Process Deployer window. The Server Mode field now displays as Development.

**Deploying a BPEL Suitcase to a Specific Domain**

In addition to the domain deployment methods described in “Compiling and Deploying a BPEL Process” on page 19-1, there are other methods for deploying a BPEL suitcase into a domain.
Managing Processes in Oracle BPEL Control

1. If the domain is local, configure the deploy option of the bpelc ant task to perform local deployment to a specific domain. The following example shows an ant build script deploying the BPEL suitcase to a domain named qa:

```xml
<?xml version="1.0"?>
<project name="LoanFlow" default="main" basedir=".">
  <property name="deploy" value="qa"/>
  <property name="rev" value="1.0"/>
  <target name="main">
    <bpelc orabpelhome="${orabpelHome}" rev="${rev}" deploy="${deploy}"/>
  </target>
</project>
```

2. If the domain is not local to the environment in which to compile the BPEL suitcase, use the Deploy New Process link under the Dashboard tab in Oracle BPEL Control to remotely upload and deploy a BPEL JAR file. Links to this task are located in the bottom-left portion of the Dashboard tab and bottom-left portion of the BPEL Processes tab. You can simply run the bpelc task without the deploy option to create the BPEL suitcase JAR in the current directory. If you have already deployed the BPEL suitcase locally, you can upload it from your local deployment directory. See “BPEL Suitcase JAR File” on page 19-5 for more information on where deployed BPEL suitcases can be found.

3. Deploying a BPEL process is equivalent to copying the BPEL suitcase JAR file into the deploy directory of the appropriate BPEL domain. Even if you are accessing the domain remotely, all you need is disk sharing, FTP, secure copy (SCP), or some other access to the server hosting the domain. You can add this to your ant build.xml script to remove the deploy option as described above and then add your own task to perform the remote copy of the generated JAR file into the appropriate location.

See Also: “Build and Command Line Tools” on page 19-27 for additional details about ant and bpelc

Undeploying a BPEL Process from a Specific Domain

Oracle BPEL Control enables you to manage the life cycle and state of a deployed BPEL process. Select the name of the BPEL process on the Dashboard tab and then select the Manage tab on the left. On this page you can first retire and then undeploy the selected BPEL process. Retiring a process prevents any new instances of that process from being created. If a specific version of a BPEL process is undeployed before all pending in-flight instances are completed, those instances are marked as stale and their execution is cancelled. Note that every task that can be performed in Oracle BPEL Control can also be performed programmatically.

Managing Processes in Oracle BPEL Control

If compilation and deployment are successful, you can run and manage the BPEL process from Oracle BPEL Control. This section provides an overview of the main pages of Oracle BPEL Control.

1. Log into Oracle BPEL Control by selecting Start > All Programs > Oracle > Oracle_Home > Oracle BPEL Process Manager > BPEL Control.

2. Enter the oc4jadmin username and password.

3. See the following sections for an overview of Oracle BPEL Control:
   - Dashboard Tab: Viewing Deployed, Running, and Completed Processes

19-12  Oracle BPEL Process Manager Developer's Guide
Managing Processes in Oracle BPEL Control

- BPEL Processes Tab: Managing the Process Life Cycle
- Instances Tab: Viewing Process Instances
- Activities Tab: Viewing Process Activities

Dashboard Tab: Viewing Deployed, Running, and Completed Processes

When you log into Oracle BPEL Control, the Dashboard tab displays by default. This page displays the currently deployed BPEL processes and instances of BPEL processes that are currently running (in-flight) and that have recently completed. Click a deployed BPEL process in the Name column to access a page for creating an instance and testing your process. Use Oracle BPEL Control to view any currently running BPEL processes before compiling and deploying additional processes. An asterisk identifies the version that is the default process. Default processes are described later in this chapter.

Viewing and Changing Domains

Each Oracle BPEL Control window includes links in the upper right corner for managing BPEL domains, accessing the BPEL site on the Oracle Technology Network, and switching to another domain. The domain into which you are currently logged is always displayed. When Oracle BPEL Process Manager is installed, an initial domain named default is created. You can create additional domains. A drop-down list enables you to access any of these domains.

1. Click the Domain list to display a list of available domains.
2. Select an appropriate domain to access (for this example, sales).

   The Dashboard tab of the selected domain appears without prompting you to enter the password.

### BPEL Processes Tab: Managing the Process Life Cycle

1. Click the BPEL Processes tab to view BPEL process life cycles and states. Note that different version labels of OrderBooking are currently active. A process identified with an asterisk (for this example, OrderBooking version 1.5) is the default process.

#### Instructions for using the sections of the BPEL Processes tab

|-----------------------|------------------|--------------------|-------------------------|---------------------|------------------|--------------------|

Instructions for using the sections of the BPEL Processes tab are listed in Table 19–2.

**Table 19–2 BPEL Processes Tab**

<table>
<thead>
<tr>
<th>For This Section...</th>
<th>See...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clear WSDL Cache</td>
<td>&quot;Clearing the WSDL Cache&quot; on page 19-14</td>
</tr>
<tr>
<td>Deploy New Process</td>
<td>&quot;Deploying New Processes&quot; on page 19-14</td>
</tr>
<tr>
<td>Refresh Alarm Table</td>
<td>&quot;Refreshing the Alarm Table&quot; on page 19-15</td>
</tr>
<tr>
<td>View Process Log</td>
<td>&quot;Viewing the Process Logs&quot; on page 19-15</td>
</tr>
<tr>
<td>Deployed Processes</td>
<td>&quot;Managing the Process Life Cycle&quot; on page 19-15</td>
</tr>
</tbody>
</table>

### Clearing the WSDL Cache

Click Clear WSDL Cache to clear the cache for all WSDLs of the selected domain.

### Deploying New Processes

Click Deploy New Process to deploy BPEL processes from Oracle BPEL Control instead of using Oracle JDeveloper.
Performing Manual Recovery
Click Perform Manual Recovery to perform a manual recovery of messages. For example, if you are using the file adapter and your system server crashes while inbound messages are being processed, you can manually perform recovery when the server restarts to ensure that all message records are recovered. For example, a file has ten messages and the server crashes after three messages have been processed. This causes the fourth message to go undelivered. When the server restarts and begins processing with message five (the offset of the last successfully rejected message), you can manually recover message four to ensure that all messages are preserved.

Refreshing the Alarm Table
Click Refresh Alarm Table to refresh the alarm table for the selected domain. This registers all pending wait/onAlarm activities with the system.

Viewing the Process Logs
Click View Process Log to view the events of all BPEL processes in the selected domain (for example, when a process was compiled, undeployed, marked as the default instance, and so on).

Managing the Process Life Cycle
This section describes how to manage the life cycle of a process.
1. Click a specific process in the BPEL Process list.
   The Manage window appears. This window enables you to manage the life cycle and state of the BPEL process.
Managing Processes in Oracle BPEL Control

For each BPEL process, Oracle BPEL Control shows the following status indicators:

- **Life cycle**

Instructions for using the sections of this window are listed in Table 19–3.

### Table 19–3 Managing the Process Life Cycle

<table>
<thead>
<tr>
<th>For This Section...</th>
<th>See...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manage</td>
<td>“Status Indicators for BPEL Processes” on page 19-16</td>
</tr>
<tr>
<td></td>
<td>“Process Life Cycle Recommendations for a Development Environment” on page 19-18</td>
</tr>
<tr>
<td></td>
<td>“Process Life Cycle Recommendations for a Production Environment” on page 19-18</td>
</tr>
<tr>
<td></td>
<td>“Example: Life Cycle of Processes” on page 19-19</td>
</tr>
<tr>
<td>Initiate</td>
<td>“Initiating Processes” on page 19-25</td>
</tr>
<tr>
<td>Descriptor</td>
<td>“Viewing and Setting Deployment Descriptors” on page 19-25</td>
</tr>
<tr>
<td>WSDL</td>
<td>“Viewing WSDL File Contents” on page 19-25</td>
</tr>
<tr>
<td>Sensors</td>
<td>“Viewing Sensor Data” on page 19-25</td>
</tr>
<tr>
<td>Source</td>
<td>“Viewing BPEL File Contents” on page 19-25</td>
</tr>
<tr>
<td>Test Suites</td>
<td>“Running Test Suites” on page 19-25</td>
</tr>
<tr>
<td>Reports</td>
<td>“Creating Reports” on page 19-25</td>
</tr>
</tbody>
</table>

**Status Indicators for BPEL Processes**

- **Life cycle**
A process life cycle can be active or retired. If the process life cycle is retired, you cannot create a new instance.

- **State**
  A process state can be on or off. If the process state is off, you cannot access instances or create new ones.

- **Open Instances**
  The number of open instances. An open instance is an instance that is currently being processed.

- **Completed Instances**
  The number of completed instances. A completed instance is an instance that has completed processing, either successfully or due to an error.

1. Perform the following process management tasks from this window:
   - Manage the process life cycle (either **Active** or **Retired**)
   - Manage the process state (either **On** or **Off**)
   - Explicitly change the default process
   - Undeploy the process

2. Ensure that you understand the following process life cycle and state concepts:

<table>
<thead>
<tr>
<th>Process Life Cycles</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Active</strong></td>
<td>All processes when deployed are automatically active (that is, existing versions are not automatically retired). You must explicitly retire processes.</td>
</tr>
<tr>
<td><strong>Retired</strong></td>
<td>A process that is no longer used. When a process is retired, all currently executing instances complete normally. You can view previously completed instances.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Process States</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>On</strong></td>
<td>Process instances can be instantiated and accessed.</td>
</tr>
<tr>
<td><strong>Off</strong></td>
<td>Process instances cannot be instantiated and accessed. Access to existing instances and activities of the process is not allowed.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Default Revision</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>A designated process and revision that is instantiated when a new request comes in.</strong></td>
<td>The default process is identified by an asterisk next to its name in Oracle BPEL Control. There can be only one default process.</td>
</tr>
</tbody>
</table>

- If you retire a default process, the default does not change to another process. The retired process remains the default. You must explicitly select a new default process.

Designating a process as the default works as follows from Oracle JDeveloper:
- **Deploy version 1.0 of the CreditRatingService process; it displays as the default process in Oracle BPEL Control.**
- **Deploy version 2.0 of the CreditRatingService process; it now displays as the default process in Oracle BPEL Control.**
- **Redeploy version 1.0 of the CreditRatingService process; it again displays as the default process in Oracle BPEL Control.**
Managing Processes in Oracle BPEL Control

Process Life Cycle Recommendations for a Development Environment

In a development environment, Oracle recommends that you always deploy processes to the same version on Oracle BPEL Server. This way, you do not need to be concerned about marking processes explicitly as default. The life cycle to follow for this environment is as follows:

- Design your process.
- Deploy the process to Oracle BPEL Server (version is 1.0). This becomes the default process for any new instances.
- Redesign the process as needed.
- Redeploy the process as version 1.0 (this is a newer version that overwrites the older version, but version 1.0 remains the default process).

Process Life Cycle Recommendations for a Production Environment

In a production environment, Oracle recommends that you increment version numbers as you deploy newer versions. For example, if OrderBooking version 1.0 is running in a production environment, then deploy the newer version of OrderBooking to version 2.0. It is your decision as to when to mark a process as default; new instances are started using this definition. When you are certain that you have adequately tested and verified your process, select Mark as Default on the Manage window shown in Step 1 on page 19-15. All version 1.0 instances switch seamlessly to version 2.0. This enables you to decide when a process is ready for production mode. The life cycle to follow for this scenario is as follows:

- Design your process.
- Deploy the process to Oracle BPEL Server with a different version number (for example, use version 2.0 if the older default version is 1.0).
- Test version 2.0 of the process.
- Activate version 2.0 by marking it as the default process.

Undeployed

A process with all traces removed from the system. You cannot view previously completed processes. Instances belonging to this process are usually purged before undeploying a process. Undeploying the only version of a process (which is also the default) results in the complete removal of this process.

If you cannot successfully undeploy a BPEL process from the Manage window of the BPSEL Processes tab of Oracle BPEL Control, then manually delete its JAR files. For example, if the process is named OrderBooking, perform the following steps:

1. Delete the following files and directories:
   - JDev_Oracle_Home\jdev\mywork\application_name\process_name\output\bpel_OrderBooking_*.jar files (for example, bpel_OrderBooking_1.0.jar, bpel_OrderBooking_2.0.jar, and so on)
   - SOA_Oracle_Home\bpel\domains\domain_name\tmp\bpel_OrderBooking_*.jar directories (for example, bpel_OrderBooking_1.0.jar, bpel_OrderBooking_2.0.jar, and so on)
2. Restart Oracle BPEL Server.

<table>
<thead>
<tr>
<th>Process</th>
<th>Description</th>
</tr>
</thead>
</table>
| Undeployed | A process with all traces removed from the system. You cannot view previously completed processes. Instances belonging to this process are usually purged before undeploying a process. Undeploying the only version of a process (which is also the default) results in the complete removal of this process. If you cannot successfully undeploy a BPEL process from the Manage window of the BPSEL Processes tab of Oracle BPEL Control, then manually delete its JAR files. For example, if the process is named OrderBooking, perform the following steps:  

1. Delete the following files and directories:  
   - JDev_Oracle_Home\jdev\mywork\application_name\process_name\output\bpel_OrderBooking_*.jar files (for example, bpel_OrderBooking_1.0.jar, bpel_OrderBooking_2.0.jar, and so on)  
   - SOA_Oracle_Home\bpel\domains\domain_name\tmp\bpel_OrderBooking_*.jar directories (for example, bpel_OrderBooking_1.0.jar, bpel_OrderBooking_2.0.jar, and so on)  
2. Restart Oracle BPEL Server. |
Example: Life Cycle of Processes This section provides a brief example of the various life cycle states of two process versions. In the first stage, two instances of the same process version are created, as shown in Table 19–4. CreditRatingService version 1.0 receives two messages, which results in the creation of two instances.

<table>
<thead>
<tr>
<th>Stage</th>
<th>Action</th>
<th>Life Cycle State</th>
<th>Default Process</th>
<th>On Arrival of New Message Request</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Deploy CreditRatingService version 1.0</td>
<td>Active=1.0, On=1.0</td>
<td>Version 1.0 (automatically set as default version in Oracle JDeveloper)</td>
<td>Create two instances for CreditRatingService version 1.0</td>
</tr>
</tbody>
</table>

The life cycle and state of the CreditRatingService version displays in the BPEL Processes tab shown in Figure 19–1. Because CreditRatingService version 1.0 was the first deployed version of this process, it automatically became the default process. The two messages that resulted in the creation of two CreditRatingService version 1.0 instances have both completed.

![Figure 19–1 Stage 1: Two Instances Created](image)

The two completed instances of CreditRatingService version 1.0 display in the Instances tab shown in Figure 19–2.

![Figure 19–2 Stage 1: Two Instances Created](image)

In stage 2, you deploy CreditRatingService again, but this time with a new version number of 2.0, as shown in Table 19–5.

**WARNING:** Do not overwrite existing versions of a process with newer versions in a production environment. This marks all existing instances of the overwritten process as stale. Stale instances cannot be examined, and all flow and audit information is lost. Instead, create a separate version as described in this section and mark the newer version as the default.
This causes CreditRatingService version 2.0 to become the default version, as indicated by the asterisk in Figure 19–3. CreditRatingService version 1.0 continues to be deployed. This is the convention followed by Oracle JDeveloper.

If you again deploy CreditRatingService in Oracle JDeveloper, and select version 1.0 in the Your version field of the Deployment Properties window, CreditRatingService version 1.0 again becomes the default version, as shown in Table 19–6 and Figure 19–4.

In stage 3, you explicitly change CreditRatingService version 2.0 in Oracle BPEL Control to be the default version and retire CreditRatingService version 1.0, as shown in Table 19–7.

<table>
<thead>
<tr>
<th>Stage</th>
<th>Action</th>
<th>Life Cycle</th>
<th>State</th>
<th>Default Process</th>
<th>On Arrival of New Message Request</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Deploy CreditRatingService version 2.0</td>
<td>Active=1.0 On=1.0</td>
<td></td>
<td>Version 2.0 (automatically set as default version in Oracle JDeveloper)</td>
<td>--</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Stage</th>
<th>Action</th>
<th>Life Cycle</th>
<th>State</th>
<th>Default Process</th>
<th>On Arrival of New Message Request</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Redeploy CreditRatingService version 1.0</td>
<td>Active=1.0 On=1.0</td>
<td></td>
<td>Version 1.0 (automatically set as default version in Oracle JDeveloper)</td>
<td>--</td>
</tr>
</tbody>
</table>

### Table 19–5 Stage 2: Multiple Process Versions Created

<table>
<thead>
<tr>
<th>Staged Action</th>
<th>Life Cycle</th>
<th>State</th>
<th>Default Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deploy CreditRatingService version 2.0</td>
<td>Active=1.0 On=1.0</td>
<td></td>
<td>Version 2.0 (automatically set as default version in Oracle JDeveloper)</td>
</tr>
</tbody>
</table>

### Table 19–6 Stage 2: Multiple Process Versions Created

<table>
<thead>
<tr>
<th>Staged Action</th>
<th>Life Cycle</th>
<th>State</th>
<th>Default Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>Redeploy CreditRatingService version 1.0</td>
<td>Active=1.0 On=1.0</td>
<td></td>
<td>Version 1.0 (automatically set as default version in Oracle JDeveloper)</td>
</tr>
</tbody>
</table>

### Figure 19–3 Stage 2: Multiple Process Versions Created

### Figure 19–4 Stage 2: Multiple Process Versions Created
Figure 19–5 shows Mark as Default being selected for CreditRatingService version 2.0. This makes it the default process.

Figure 19–6 shows CreditRatingService version 1.0 being retired.

### Table 19–7 Stage 3: Change Default Process and Retire Instance

<table>
<thead>
<tr>
<th>Stage</th>
<th>Action</th>
<th>Life Cycle</th>
<th>State</th>
<th>Default Process</th>
<th>On Arrival of New Message Request</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Change default process to OrderBooking version 2.0</td>
<td>Active=1.0 On=1.0</td>
<td>Version 2.0 (explicitly set in Oracle BPEL Control)</td>
<td>Create an instance for OrderBooking version 2.0</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Retire OrderBooking version 1.0</td>
<td>Retired=2.0 On=2.0</td>
<td>Version 2.0</td>
<td>No action</td>
<td></td>
</tr>
</tbody>
</table>

Figure 19–5 Stage 3: Change Default Process and Retire Instance

<table>
<thead>
<tr>
<th>BPEL Process: CreditRatingService Version: 2.0 Lifecycles: Active</th>
<th>Dashboard</th>
<th>BPEL Processes</th>
<th>Instances</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manage</td>
<td>Indicate</td>
<td>Simulator</td>
<td>NSDL</td>
</tr>
</tbody>
</table>

**Process Lifecycle:** When the process lifecycle is **active**, instances may be instantiated from the process; when it is **retired**, no new instances may be instantiated but existing instances are permitted to complete normally.

- **Active**
- **Retired**

**Process State:** The process state controls overall access to the process. When the state is **on**, instanciation is permitted and access to existing instances and activities belonging to the process will be enabled. When the state is **off**, no new instances may be instantiated and access to existing instances and activities belonging to the process will be disabled.

- **On**
- **Off**

**Default Revision:** The default revision is selected if an invocation message does not specify the revision tag of the process to instantiate.

- **Mark as Default**
The modified life cycle and state of the two CreditRatingService versions displays in the BPEL Processes tab shown in Figure 19–7. Because CreditRatingService version 2.0 was explicitly selected as the default process, it now displays the asterisk. The message that resulted in the creation of an CreditRatingService version 1.0 instance has completed. CreditRatingService version 1.0 displays as Retired.

The completed instance of CreditRatingService version 2.0 displays in the Instances tab shown in Figure 19–8.
If you click the **Dashboard** tab, the retired **CreditRatingService** version 1.0 no longer appears. This means you can no longer create an instance for this version.

In stage 4, you make **CreditRatingService** version 1.0 inactive and then undeploy it, as shown in Table 19–8.

<table>
<thead>
<tr>
<th>Stage</th>
<th>Action</th>
<th>Life Cycle</th>
<th>State</th>
<th>Default Process</th>
<th>On Arrival of New Message Request</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>Make</td>
<td>Retired=2.0</td>
<td>Off=2.0</td>
<td>Version 2.0</td>
<td>No action</td>
</tr>
<tr>
<td></td>
<td>CreditRatingService version 1.0 inactive</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Undeploy</td>
<td>Retired=2.0</td>
<td>Off=2.0</td>
<td>Version 2.0</td>
<td>No action</td>
</tr>
<tr>
<td></td>
<td>CreditRatingService version 1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The state of **CreditRatingService** version 1.0 is changed to **Off** (inactive) in Figure 19–9.

**Figure 19–9**  **Stage 4: Deactivate and Undeploy a Process**

The state of **CreditRatingService** version 1.0 displays as **Off** in the **BPEL Processes** tab shown in **Figure 19–10**. Because you initially retired this process, any live instances had already completed normally. If you had instead made this version inactive before retiring it, any live instances would have faulted and been aborted.
CreditRatingService version 1.0 is then undeployed, as shown in Figure 19–11. The BPEL Processes tab in Figure 19–12 no longer displays CreditRatingService version 1.0. The asterisk also no longer displays for CreditRatingService version 2.0. However, this version is still the default. If you click this instance in the BPEL Process list, you see that no Mark as Default button displays in the Manage window. Instead, the following message appears.

This revision is currently selected as the default revision.

The two completed instances of the undeployed CreditRatingService version 1.0 display as disabled in the Instances tab shown in Figure 19–13.

Clicking one of the completed instances displays the status as Stale in Figure 19–14.
Figure 19–14  Stage 4: Deactivate and Undeploy a Process

The state of this instance is Stale.

The following actions may be applied to the instance:

[ ] Deactivate the instance

Initiating Processes  Click **Initiate** to run processes from the BPEL Processes tab. This is the same window that displays when you click a process in the Deployed BPEL Processes list of the Dashboard tab.

Viewing and Setting Deployment Descriptors  Click **Descriptor** to view and change deployment descriptor **bpel.xml** file properties of a BPEL process at run time. This prevents you from having to reset these properties during design time and redeploy the BPEL process.

See Also:  Appendix C, “Deployment Descriptor Properties”

Viewing WSDL File Contents  Click **WSDL** to view the WSDL file contents for a process.

Viewing Sensor Data  Click **Sensors** to view the fault, activity, and variable sensor data of a process.

See Also:  “Viewing Sensor and Sensor Action Definitions” on page 17-11

Viewing BPEL File Contents  Click **Source** to view the BPEL file contents of a process.

Running Test Suites  Test suites enable you to simulate the interaction between a BPEL process and its Web service partners prior to deployment in a production environment. This helps to ensure that a process interacts with Web service partners as expected by the time it is ready for deployment to a production environment. Click **Test Suites** to run the test cases of a deployed test suite for a BPEL process instance and view XML document reports. By default, report results are formatted as JUnit XML test results.

See Also:  Chapter 20, “Testing BPEL Processes”

Creating Reports  Click **Reports** to create reports in Oracle BPEL Control that enable you to:

- Receive an overall view of business process instance performance
- Analyze data for the BPEL process instances and make critical decisions
- Analyze data of the activities that constitute a business process
- Identify and debug faults and take appropriate corrective actions

See Also:  Chapter 22, “Oracle BPEL Control Reports”
Instances Tab: Viewing Process Instances

1. Click the Instances tab to view BPEL process instances.

2. Click an instance in the Instance column (for example, Instance #30 of OrderBooking). From the window that appears, you can perform the following tasks:
   - View the state of the instance (for example, Completed, Active, or Faulted)
   - Delete the instance.
   - Click Flow to view a visual representation of the history of the activities in this instance.
   - Click Audit to view an audit trail of this instance.
   - Click Debug to view the BPEL Debugger, which takes the BPEL source code that implements this process and matches it against the state of this particular instance. Points in the code where execution is currently paused are highlighted in yellow (for example, the process is currently waiting for a loan service to call back with a loan offer).
   - Click Interactions to view details about the activities in this instance.
   - Click Sensor Values to view the results of any activity, fault, or variable sensors you created in this instance.
   - Click Test to save an instance as a test case. You can then import it into an Oracle JDeveloper project.

   **Note:** The Flow and Audit links do not provide details about the actions or states of any header variables you defined during design time. As a workaround, use the Debug link to view header variable details.
Activities Tab: Viewing Process Activities

1. Click the Activities tab to view the status of activities in the deployed BPEL process instance.

Build and Command Line Tools

Build and Command Line Tools

See Also: The following documentation for additional details about sensors and test cases:

- Chapter 17, “Sensors”
- Chapter 20, “Testing BPEL Processes”
- Oracle BPEL Process Manager Order Booking Tutorial

See Also: The following documentation for tutorials in which you run processes from Oracle BPEL Control and view their results from the Audit and Flow links:

- Oracle BPEL Process Manager Order Booking Tutorial
- Oracle BPEL Process Manager Quick Start Guide

Build and Command Line Tools

When you deploy a BPEL process, several build and compiler command line tools are automatically invoked. This section provides an overview of these tools, plus an additional command line tool for generating XML facades:

- ant
- bpelc
- schemac
ant

ant is a Java-based build tool used by Oracle BPEL Process Manager for compiling and deploying the BPEL process. ant is similar to a make file. However, instead of being extended with operating system-dependent, shell-based commands, ant is extended using Java classes. The configuration files are XML-based and call out a target tree where various tasks are executed.

See Also:  http://ant.apache.org/

bpelc

bpelc (or bpelc.sh for UNIX operating systems) is the Oracle BPEL Process Manager tool that compiles and deploys BPEL processes.

Table 19–9 shows the supported bpelc options:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>classpath</td>
<td>Specifies where to find user class files. This attribute is similar to a PATH structure and can also be set through a nested classpath element.</td>
<td>No</td>
</tr>
<tr>
<td>deploy</td>
<td>Deploys the BPEL process archive to the specified domain in the local Oracle home. The domain must be accessible through the file system for this option to work.</td>
<td>No</td>
</tr>
<tr>
<td>force</td>
<td>Always compiles the process; the compiler does not check the time stamp of .bpel, .wadl and .xml files. The value defaults to false.</td>
<td>No</td>
</tr>
<tr>
<td>help</td>
<td>Displays the help message. This value defaults to false.</td>
<td>No</td>
</tr>
<tr>
<td>home</td>
<td>The orabpel home directory (or whatever you named your SOA_Oracle_Home directory), which is typically available as ant property home.</td>
<td>No</td>
</tr>
<tr>
<td>input</td>
<td>The deployment descriptor location path, By default, it looks for bpel.xml under the current directory.</td>
<td>No</td>
</tr>
<tr>
<td>keepGenerated</td>
<td>Includes the BPEL process Java classes in the generated BPEL archive. The value defaults to false.</td>
<td>No</td>
</tr>
<tr>
<td>lib</td>
<td>Oracle BPEL Process Manager system lib directory.</td>
<td>No</td>
</tr>
<tr>
<td>out</td>
<td>Specifies the location in which to deploy the BPEL archive. This option is used when the deploy attribute is not used. For example: out=&quot;c:\myproject\bpel\deploy&quot;</td>
<td>No</td>
</tr>
<tr>
<td>rev</td>
<td>The revision (version) tag for the deployed process.</td>
<td>No</td>
</tr>
<tr>
<td>verbose</td>
<td>Generates additional debugging messages about compiler actions. The value defaults to false.</td>
<td>No</td>
</tr>
</tbody>
</table>

Examples of ant Tasks

The following ant task compiles and generates a BPEL archive file in the current directory using the default bpel.xml deployment descriptor.

Use the following bpelc task sample to deploy a BPEL archive into the default domain deploy directory:

```xml
<bpelc home="${home}" rev="${rev}" deploy="default"/>
```
To deploy the BPEL archive into the c:\myproject directory:

```xml
<bpelc home="${home}" rev="${rev}" out="C:\myproject"/>
```

Specify a deployment descriptor file name:

```xml
<bpelc home="${home}" rev="${rev}" deploy="default" input="orderdd.xml"/>
```

Specify a user classpath for bpelc:

```xml
<bpelc home="${home}" rev="${rev}" deploy="default"/>
<classpath>
<path element location="dist/test.jar"/>
<path element path="${java.class.path}"/>
</classpath>
</bpelc>
```

`schemac` (or `schemac.sh` for UNIX operating systems) is a schema compiler utility provided with Oracle BPEL Process Manager. You use this utility to generate XML facades. XML facades are a set of Java interfaces and classes through which you can access and modify BPEL (and other XML) variables and map individual XML values to Java variables with `get` and `set` methods. Classes are generated only for the `complexType` schema types and element with an anonymous `complexType`. This is similar to the `jaxb` compiler.

You can invoke `schemac` from the operating system command prompt to perform specific tasks. `schemac` command line syntax uses the following format:

```
`schemac options filename | classname(s)
```

where `filename` is the name of a file ending with `.xsd` and containing a valid XML schema definition and `classname` is the name of a valid Java class (without the `.java` suffix). Only use this argument when the `-R` option is supplied.

Table 19–10 describes the supported options:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>input</td>
<td>The XML schema is the name of a file (ending with .xsd or .wsdl) containing a valid XML schema definition.</td>
<td>Yes</td>
</tr>
<tr>
<td>out</td>
<td>Specify where to place generated facade class files. This value defaults to the current directory.</td>
<td>No</td>
</tr>
<tr>
<td>doc</td>
<td>Generates Javadoc for the generated classes and redirects it to the specified location.</td>
<td>No</td>
</tr>
<tr>
<td>jar</td>
<td>Archives the generated classes into the specified JAR file name.</td>
<td>No</td>
</tr>
<tr>
<td>verbose</td>
<td>Generates more debugging messages about the compiler actions.Defaults to <code>false</code>.</td>
<td>No</td>
</tr>
</tbody>
</table>

Note: While `schemac` is currently included with Oracle BPEL Process Manager, Oracle recommends that you use JAXB, which provides a standard Java object-to-XML API. See the Oracle Application Server TopLink Application Developer’s Guide for details about JAXB. `schemac` will not be included with Oracle Application Server 11.
Examples

Generate the facade classes from an XSD and place them under the current directory:
<schemac input="${basedir}/Order.xsd"/>

Generate the facade classes from a WSDL schema file:
<schemac input="${basedir}/PurchaseOrder.wsdl"/>

Generate the Javadoc into the c:\myjavadoc directory:
<schemac input="${basedir}/Order.xsd" doc="c:\myjavadoc"/>

Archive the generated facade classes into a .jar file:
<schemac input="${basedir}/XPath.wsdl" jar="myorderfacade.jar"/>

Redirect the generated facade classes into a specific directory:
<schemac input="${basedir}/Order.xsd" out="${basedir}/BPEL-INF/classes"/>

Specify the namespace Java package mapping file to override the default behavior:
<schemac input="${basedir}/Order.xsd" nsMap="mynsmap.txt" noCompile="false" doc="c:\myjavadoc" jar="myorderfacade.jar" out="${basedir}/BPEL-INF/classes"/>

Table 19–10 (Cont.) Parameters

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>noCompile</td>
<td>Generates only the Java source files and does not compile the generated sources when set to true. This value defaults to false.</td>
<td>No</td>
</tr>
<tr>
<td>help</td>
<td>Displays the help message. This value defaults to false.</td>
<td>No</td>
</tr>
<tr>
<td>sourceOut</td>
<td>Specifies the location in which to redirect the generated Java files. For example: sourceOut=&quot;c:\myproject\bpe\facade\source&quot;</td>
<td>No</td>
</tr>
</tbody>
</table>
| nsMap | To override the default Java package name, specify the namespace to the Java package mapping file. For example:
nsMap="mynsmap.txt"

The content of mynsmap.txt looks as follows:
http://samples.otn.com/xpath/autoloan=boo.foo

Note: If it is a name-value property file, you must escape the colon (:) using a backslash (\). If there is no nsMap attribute, by default schemac generates the package name from the namespace. For example, the default Java package name for http://samples.otn.com/xpath/autoloan is com.otn.samples.xpath.autoloan.
This chapter describes how to compile and deploy BPEL processes. It describes key features of BPEL suitcase JAR files. It also describes how to create and manage BPEL domains, including creating domains, changing Oracle BPEL Server modes (production or development), managing BPEL suitcase JAR files, and undeploying processes. An overview of Oracle BPEL Control is also provided, including a detailed description of managing different versions of BPEL processes. Finally, a discussion on how to use the ant, bpelc, and schemac build tools is provided.

See Also:
- Oracle Application Server TopLink Application Developer’s Guide for details about JAXB
- "Using an XML Facade to Simplify DOM Manipulation" on page 9-7
- SOA_Oracle_Home\bpeI\samples\tutorials\702Bindings for XML facade samples

Summary
This chapter describes how to create, deploy, and run test cases that automate the testing of BPEL processes. Test cases enable you to simulate the interaction between a BPEL process and its Web service partners prior to deployment in a production environment. This helps to ensure that a process interacts with Web service partners as expected by the time it is ready for deployment to a production environment.

This chapter contains the following topics:

- Overview of the BPEL Test Framework
- Components of a Test Suite
- Creating Test Suites in Oracle JDeveloper
- Deploying a Test Suite
- Running a Test Suite and Viewing Report Results
- Advanced Test Suite Design Features

See Also: BPEL test suite sample files located at:

- SOA_Oracle_Home\bpel\samples\references\BPELTest
- SOA_Oracle_Home\bpel\samples\demos\BPELTest

**Overview of the BPEL Test Framework**

Oracle BPEL Process Manager provides an automated test suite framework for creating and running repeatable tests on a BPEL process. The test suite framework provides the following features:

- Simulates Web service partner interactions
- Validates process actions with test data
- Calculates the percentage of source code executed in terms of the percentage of simple activities executed
- Generates a test case from the audit trail of a completed instance
- Creates reports of test results

The following sections provide an overview of test suite concepts:

- Test Cases Overview
- Test Suites Overview
- Emulations Overview
Overview of the BPEL Test Framework

- Assertions Overview
- Process Code Coverage Overview
- JUnit Support Overview

Test Cases Overview

The test framework supports two types of test cases:

- Unit test — Represents a single test case in a test suite. For example, assume you have a BPEL process in which an offer for a product is submitted to two Web service suppliers. You can create a test in your test suite to emulate the behavior of these Web service partners in this interaction with your BPEL process.

- Composite test — Consists of a test where BPEL partners are not emulated, but instead are invoked with test case information. For example, assume you have a BPEL process that calls a subprocess that contains a human workflow step. You may want to test the interactions between the two BPEL processes, but not have to manually perform the human workflow approval step. In this case, you can create a composite test for the main process by passing a test case name to the subprocess when it is invoked. This test case can emulate the human workflow step; this means that no human interaction is required.

See Also:  “Creating Test Suites in Oracle JDeveloper” on page 20-6

Test Suites Overview

Test suites consist of a logical collection of one or more test cases. Each test case contains a set of commands to perform as the test instance is executed. The execution of a test suite is known as a test run. Each test corresponds to a single BPEL instance.

See Also:
- “Creating Test Suites in Oracle JDeveloper” on page 20-6
- “Editing Test Cases in Oracle JDeveloper” on page 20-9

Emulations Overview

Emulations enable you to simulate the behavior of Web service partners with which your BPEL process interacts during execution. Instead of invoking a partner link, you can specify a response.

See Also:
- “Emulations” on page 20-4
- “Creating Emulations in Oracle JDeveloper” on page 20-10

Assertions Overview

Assertions enable you to verify variable data or process flow. You can perform the following types of assertions:

- Simple Value Assert — Compare the value of a selected string or number variable to an expected value. An error message that you define is displayed if a comparison fails.
Components of a Test Suite

- **XML Assert** — Compare the element values of an entire XML document to the expected element values. An error message that you define is displayed if a comparison fails.
- **Activity Executed Assert** — Execute an activity a specified number of times. This ensures that an activity executes the correct number of times. This is useful for verifying process flow.

**See Also:**
- "Assertions" on page 20-5
- "Creating Assertions in Oracle JDeveloper" on page 20-14

Process Code Coverage Overview

Code coverage provides a method for calculating the completeness of the executed tests. This is calculated as the percentage of simple activities executed at least once, compared to the number of simple activities defined in the BPEL process. Simple activities are nonstructured activities such as invoke, receive, reply, and assign activities.

**See Also:** "Running a Test Suite and Viewing Report Results" on page 20-21

JUnit Support Overview

JUnit is an open source test framework to use for creating regression tests for Java applications. JUnit is an instance of the xUnit architecture for unit testing frameworks. By default, the reports created during test suite execution are in JUnit XML format.

**Note:** While reports display in JUnit XML format, the tests from which reports are created are Oracle BPEL Process Manager tests, and not JUnit tests.

**See Also:**
- http://www.junit.org
- "Running a Test Suite and Viewing Report Results" on page 20-21

Components of a Test Suite

This section describes the test components that comprise a test case. The PriceFinder demonstration is used as an example. Methods for creating and importing these tests into your process are described in subsequent sections of this chapter.

This section contains the following topics:
- Process Initiation
- Emulations
- Assertions
- Include Files
Components of a Test Suite

See Also:

- SOA_Oracle_Home\bpel\samples\demos\BPELTest\PriceFinder
- SOA_Oracle_Home\bpel\samples\references\BPELTest

Process Initiation

You first define the operation of your process. The following section defines the operation of `initiate` to initiate the PriceFinder process. The initiation payload is also defined in this section:

```xml
<BPELTest processName="PriceFinderWithTests" xmlns="http://xmlns.oracle.com/bpel/instancedriver">
  <initiate operation="initiate">
    <inboundMessage>
      <part name="payload">
        <PriceProviderProcessRequest xmlns="http://xmlns.oracle.com/PriceProvider">
          <manufacturer>Oracle</manufacturer>
          <ItemName xmlns="http://xmlns.oracle.com/PriceProvider">BPEL PM</ItemName>
          <customerLocation xmlns="http://xmlns.oracle.com/PriceProvider">94065</customerLocation>
        </PriceProviderProcessRequest>
      </part>
    </inboundMessage>
  </initiate>
</BPELTest>
```

Emulations

You create emulations to simulate the message data that your BPEL process receives from Web service partners. PriceFinder invokes two pricing services:

- FlakyPriceProvider
- FreeShippingPriceProvider

The following test code instructs Oracle BPEL Process Manager to first skip the outbound invocation to the `FreeShippingPriceProvider` service and then emulate receiving a response:

```xml
<BPELTest processName="LoanFlow" processRevision="1.0" xmlns="http://xmlns.oracle.com/bpel/instancedriver">
  <!-- *****************************************
  Skip outbound invoke
  ***************************************** -->
  <activityDriver name="FreeShippingInvoke">
    <emulate/>
  </activityDriver>
  <!-- *****************************************
  Emulate the FreeShippingPriceProvider Service
  ***************************************** -->
  <activityDriver name="FreeShippingReceive" firstIteration="1"/>
</BPELTest>
```
In the first test, the emulate element is empty. This is because the activity is a one-way invoke, and is skipped. In the second test, a receive activity is emulated. This means you must specify the inbound message.

**See Also:**
- "Creating Emulations in Oracle JDeveloper" on page 20-10

**Assertions**

You create assertions to validate a variable or an entire XML document at a point during BPEL process execution.

This test tells Oracle BPEL Process Manager that after the choosePrice activity has completed, ensure that the item price content in the outputVariable variable matches the content specified.

**Note:** Test case content can also be created from the audit trail in Oracle BPEL Control.

**See Also:**
- "Creating Assertions in Oracle JDeveloper" on page 20-14
- "Creating a Test Case from Oracle BPEL Control" on page 20-17
Include Files

Large portions of tests typically stay the same across different test cases. To avoid having to duplicate large sections of test files, tests can include other tests and then selectively override particular tests to create various test cases. These are known as baseline tests. Baseline tests do not run on their own; they exist only to be included by other tests.

Including a baseline test in a test case brings everything from the baseline test into the test case. If a particular action in the baseline file is not needed, it can be overridden in the test case.

For example, you can define the process operation and payload information shown in “Process Initiation” on page 20-4 in a separate file named baseline.xml. The baseline test must be stored in the includes directory of your test suite in Oracle JDeveloper:

In your main test case file, you call this file as follows:

```xml
<BPELTest processName="PriceFinderWithTests"
            xmlns="http://xmlns.oracle.com/bpel/instancedriver"
            <include>
                baseline.xml
            </include>
```

See Also: “Creating Test Suites in Oracle JDeveloper” on page 20-6 for information about creating these test components

Creating Test Suites in Oracle JDeveloper

You first create a test suite in which you then create or import test cases. There are several methods for adding test cases to a test suite:

- You import test cases into a test suite. You create these test files manually or through audit trail output from Oracle BPEL Control.
- You can create a new test case using Oracle JDeveloper.

This section contains the following topics:

- Creating Test Suites in Oracle JDeveloper
- Importing Test Cases in Oracle JDeveloper
- Creating Test Cases in Oracle JDeveloper
- Editing Test Cases in Oracle JDeveloper
- Creating a Test Case from Oracle BPEL Control

Note: Do not enter a multibyte character string as a test suite name or test case name. Doing so causes an error to occur when the test is executed from Oracle BPEL Control.

Creating Test Suites in Oracle JDeveloper

This section describes how to create test suites for a BPEL process in Oracle JDeveloper.

1. Expand the BPEL process in which to create a test suite in the Application Navigator.
Creating Test Suites in Oracle JDeveloper

Testing BPEL Processes 20-7

This displays the Test Suites folder under Integration Content.

2. Right-click Test Suites and select Create Test Suite.

3. Enter a test suite name (for example, logicTest).

4. Click OK.

   The test suite is created beneath the Test Suites folder in the Application Navigator:

   The following operating system directory is also created:

   \JDev_Oracle_Home\jdev\mywork\application_name\process_name\bpel\testsuites\test_suite_name

   Two subdirectories for adding additional test files are also created beneath test_suite_name: includes and messages. These subdirectories are represented by the Includes tab and Message Instance Files tab, respectively, of the Edit Test Suite window, which is described in Step 1 of "Importing Test Cases in Oracle JDeveloper" on page 20-7.

   Note: You cannot create test suites within other test suites. However, you can organize a test suite into subdirectories.

Importing Test Cases in Oracle JDeveloper

This section describes how to import test cases into a test suite in Oracle JDeveloper.

1. Right-click the test suite name you entered in Step 3 of "Creating Test Suites in Oracle JDeveloper" on page 20-7 and select Edit Test Suite.

   This displays the Edit Test Suite window.
Creating Test Suites in Oracle JDeveloper

You can create the content for test cases manually or through audit trail output from Oracle BPEL Control. You then import the files into Oracle JDeveloper. You can validate a test case in the Application Navigator by right-clicking it and selecting Validate XML.

This window consists of the following tabs for importing test cases:

<table>
<thead>
<tr>
<th>Tab</th>
<th>Description</th>
</tr>
</thead>
</table>
| General | The tab enables you to add test cases to your test suite. Add a test file by clicking Add. These files are added to the following directory:  
JDev_Oracle_Home\jdev\mywork\application_name,  
process_name\bpel\testsuites\test_suite_name  
You can also select to add this test file as a baseline test. A baseline test is a generic test that can be imported and used by other tests. These files enable you to factor out common testing actions so they do not need to be repeated in multiple files. |
| Includes | This tab enables you to add baseline (include) test cases. Add a baseline test file by clicking Add. These files are added to the following directory:  
JDev_Oracle_Home\jdev\mywork\application_name,  
process_name\bpel\testsuites\test_suite_name\includes  
Baseline files consist of generic tests that do not run on their own. Instead, these tests are imported and called by other tests. Including a baseline test in a test case brings everything from the baseline test into the test case. If a particular action in the baseline is not desired, it can be overridden in the test case. |
Creating Test Cases in Oracle JDeveloper

1. Right-click the test suite name you entered in Step 3 of "Creating Test Suites in Oracle JDeveloper" on page 20-6 and select Create BPEL Test.

2. Enter the following details:

   3. Click OK.

   4. Select this file in the Application Navigator and click Source.

   5. Edit the file to include all necessary test details.

Creating Test Cases in Oracle JDeveloper

You can create an empty test case in your test suite.

1. Double-click a test case beneath the Test Suites folder in the Application Navigator.

The BPEL process in Oracle JDeveloper is refreshed to display in test mode. This mode enables you to define test information. No other modifications, such as editing the property windows of activities, can be performed in this node.

2. If you want to view the XML source of the test case, select the file in the Application Navigator and click Source.

3. Edit the file as necessary.

Editing Test Cases in Oracle JDeveloper

Test cases consist of emulations, assertions, and external calls. You can add these actions to test cases in the test mode of Oracle JDeveloper.

1. Double-click a test case beneath the Test Suites folder in the Application Navigator.

The BPEL process in Oracle JDeveloper is refreshed to display in test mode. This mode enables you to define test information. No other modifications, such as editing the property windows of activities, can be performed in this node.
Creating Test Suites in Oracle JDeveloper

Activities on which these actions have already been defined display special icons in the upper right corner.

2. Right-click an activity to display a list of test actions that can be performed.

3. Select one of the following menu options and see the corresponding section for details:

<table>
<thead>
<tr>
<th>Menu Option</th>
<th>See Section...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Emulate activity_type Message</td>
<td>&quot;Creating Emulations in Oracle JDeveloper&quot; on page 20-10</td>
</tr>
<tr>
<td>Asserts</td>
<td>&quot;Creating Assertions in Oracle JDeveloper&quot; on page 20-14</td>
</tr>
<tr>
<td>External Calls</td>
<td>&quot;Creating External Calls in Oracle JDeveloper&quot; on page 20-16</td>
</tr>
</tbody>
</table>

Creating Emulations in Oracle JDeveloper

You create emulations to simulate either message data, fault data, or both types that your BPEL process receives from Web service partners. The fields that display on this window are based on the activity type selected (an invoke or receive are supported) and the radio buttons selected at the top of this window:

- Emulating Inbound Messages
- Emulating Faults
- Emulating BPEL or Partner Tests
Creating Test Suites in Oracle JDeveloper

Testing BPEL Processes

20-11

Emulating Inbound Messages

Select this check box to send an inbound message from a client, then select the type of return message data to simulate from the Web service partner. This feature is available for receive activities and two-way invoke activities.

1. Select **Emulate Inbound Message**.
2. Enter the following details:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operation</td>
<td>This field is automatically completed with the operation type for the activity (for example, a process operation in an invoke activity).</td>
</tr>
<tr>
<td>Message</td>
<td>This field is automatically completed with the path to the inbound message.</td>
</tr>
<tr>
<td>Part</td>
<td>Select the part of the inbound message (for example, a payload).</td>
</tr>
<tr>
<td>Value</td>
<td>Create a simulated message to return from a Web service partner.</td>
</tr>
<tr>
<td></td>
<td>■ <strong>Enter Manually</strong> Click to enter message data in the <strong>Enter Value</strong> field.</td>
</tr>
<tr>
<td></td>
<td>■ <strong>Load From File</strong> Click the <strong>flashlight</strong> icon to load message data from a file.</td>
</tr>
<tr>
<td>Duration</td>
<td>Enter the amount of time to wait for the message to be delivered from the Web service partner. This field displays for two-way invoke activities.</td>
</tr>
</tbody>
</table>

Notes:

- One-way invokes can be skipped. This is because these types of invokes do not receive any data from the partner link.
- You can override the emulations defined in the top-level test case added in the Edit Test Suite - General tab by clicking **Override Included Emulation** in the BPEL Test Settings - Emulate tab.

See Also:

- **“Emulations Overview”** on page 20-2
- **“Emulations”** on page 20-4

An example of this window with completed content is shown below:
Emulating Faults Select this check box to send an inbound system fault from a client. Then, select the type of return fault message to simulate from the Web service partner. This enables you to test fault handling capabilities in your process. This feature is available for invoke activities.

1. Select Emulate Fault.
2. Enter the following details:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Namespace URI</td>
<td>A fault must have a unique qualified name (QName). This activity must provide a name for the fault and optionally provide a variable of data that provides further information about the fault. Click the flashlight icon to select the fault to monitor. The Namespace URI field is automatically completed with a URL path based on your fault selection.</td>
</tr>
<tr>
<td>Local Part</td>
<td>Displays the local part selection you make for the Namespace URI field. For example, if you select a Web service partner with an associated fault named NegativeCredit, then the name NegativeCredit is added to this field.</td>
</tr>
<tr>
<td>Part</td>
<td>Select the message part containing the fault (for example, a payload).</td>
</tr>
<tr>
<td>Value</td>
<td>Create a simulated fault message to return from a Web service partner:</td>
</tr>
<tr>
<td>Enter Manually</td>
<td>Click to enter message data in the Enter Value field.</td>
</tr>
<tr>
<td>Load From File</td>
<td>Click the flashlight icon to load message data from a file.</td>
</tr>
</tbody>
</table>

An example of this window with completed content is shown below:
**Emulating BPEL or Partner Tests** In the case where the partner is a BPEL process, you can pass a test case for this partner to execute. This is useful if you want to test the interaction between your BPEL processes, but not any external partners. For example, assume you have a main BPEL process that calls a subprocess that in turn calls human workflow from a test for the main BPEL process. Instead of emulating the subprocess, you can invoke a test case of the subprocess that emulates the human workflow service. This enables you to test the interoperability of your BPEL processes without having to invoke the human workflow service. The name of this radio button is based upon the type of activity selected. For one-way invoke activities, the name is **Call Partner Test**. For two-way invoke activities, the name is **Call BPEL Test**. For receive activities, this radio button is disabled.

1. Select **Call Partner Test** or **Call BPEL Test**.
2. Enter the name and relative location of the test in the test suite to run.

   An example of this window with completed content is shown below:

**Note:** For one-way invokes, only system faults can be emulated. For two-way invokes, both system and user-defined business faults can be emulated.
Creating Assertions in Oracle JDeveloper

You perform assertions to verify variable data or process flow. Variable data assertions enable you to validate test data in a variable as a process is executed. This is done by extracting a value from a variable or an XML document and comparing it to an expected value. To verify process flow, you can assert the number of times that an activity has been executed.

1. Click Create to display a list of tests.

2. Select a test and see the corresponding section for details.

<table>
<thead>
<tr>
<th>Selection</th>
<th>See Section...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value Assert</td>
<td>“Creating Value Asserts” on page 20-14</td>
</tr>
<tr>
<td>Activity Executed Assert</td>
<td>“Creating an Activity Execution Assert” on page 20-15</td>
</tr>
<tr>
<td>XML Assert</td>
<td>“Creating XML Asserts” on page 20-15</td>
</tr>
</tbody>
</table>

See Also:
- “Assertions Overview” on page 20-2
- “Assertions” on page 20-5

Creating Value Asserts

This test compares the value of a selected string or number variable to an expected value. The XPath expression specified must resolve to a simple type (for example, string, number, and so on).

1. Enter the following details:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable</td>
<td>Click the flashlight icon to select a variable XPath value. The value for this variable is extracted during testing and compared to the expected value defined below.</td>
</tr>
<tr>
<td>Part</td>
<td>Displays the part selection you make for the Variable field.</td>
</tr>
<tr>
<td>Actual Path</td>
<td>Displays the path selection you make for the Variable field.</td>
</tr>
<tr>
<td>Comparison Method</td>
<td>Select the variable value type to perform a string comparison or numerical comparison.</td>
</tr>
</tbody>
</table>

Oracle BPEL Process Manager Developer's Guide
Creating Test Suites in Oracle JDeveloper

Testing BPEL Processes 20-15

Creating an Activity Execution Assert

This test executes an activity a specified number of times. This provides a method for verifying that an activity executes the correct number of times (for example, ensuring that a while activity executes the correct number of times).

1. Enter a value for the number of times to execute this activity.

Creating XML Asserts

This test compares the element values of an entire XML document to the expected element values.

1. Enter the following details:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable</td>
<td>Click the flashlight icon to select a variable XPath value. The element value for this variable is extracted during testing and compared to the expected element value defined below.</td>
</tr>
<tr>
<td>Part</td>
<td>Displays the part selection you make for the Variable field.</td>
</tr>
<tr>
<td>Actual Path</td>
<td>(Optional) Displays the path selection you make for the Variable field.</td>
</tr>
</tbody>
</table>

Warning: For this release, fatal assertions are not supported at runtime. Create nonfatal assertions only. Test instances with fatal assertions hang and can impact Oracle BPEL Server performance.
Creating External Calls in Oracle JDeveloper

External calls provide an advanced method for performing user actions during the execution of a test case. You specify a fully qualified Java class name for the command. When the test case reaches the activity where the external call is set, the specified Java class is called with the provided arguments. This class must implement the com.oracle.services.bpel.test.ITestCallHandler interface. At run time, this class must be in the ant classpath when running the task. Call handlers are not supported when running tests from Oracle BPEL Control.

An example of this window with completed content is shown below:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comparison Method</td>
<td>Specify the strictness of the comparison. For most purposes, xml-similar is sufficient. xml-identical can be used when the comparison must be exact (for example, element ordering).</td>
</tr>
<tr>
<td>Expected Value</td>
<td>Select a method for comparing the values:</td>
</tr>
<tr>
<td></td>
<td>Select and manually enter the element value you are expecting the variable to contain.</td>
</tr>
<tr>
<td></td>
<td>Select and click the flashlight icon to choose a message file containing the element value you are expecting the variable to contain.</td>
</tr>
<tr>
<td>Error Message</td>
<td>Enter a message to display if the assertion fails.</td>
</tr>
<tr>
<td>Fatal</td>
<td>Select this check box if you want the assertion to be fatal. This causes the instance to immediately terminate. If not selected, the instance continues to execute.</td>
</tr>
</tbody>
</table>

Warning: For this release, fatal assertions are not supported at runtime. Create nonfatal assertions only. Test instances with fatal assertions hang and can impact Oracle BPEL Server performance.
Creating Test Suites in Oracle JDeveloper

1. Click Create.
   The External Call window appears.
2. Click Create to create a default argument name and value in the table.
3. Place your cursor inside the default values for the Name and Value columns, remove the default values, and enter appropriate details.
4. If you want to execute this command before this activity is executed, select the Do Before Activity check box. Otherwise, the call is executed after the activity.
   An example of this window with completed content is shown below:

5. Click OK.
   When the call is executed, these arguments are passed to the Java class.

See Also:
  - Oracle BPEL Process Manager Client API Reference:
    JSA_Oracle_Home/bpel/docs/apidocs

Creating a Test Case from Oracle BPEL Control

You can create the contents for your test case through audit trail output from Oracle BPEL Control. This provides two benefits:

- A quick method for creating a baseline test when first starting.
- A method for converting an error scenario identified during development into a test case. You capture the use case that exposed the bug, add specific data assertions, and add the test to a test suite to run regularly as a regression test. This enables you to ensure that the bug does not re-appear in the future.

1. Log into Oracle BPEL Control by selecting Start > All Programs > Oracle - Oracle_Home > Oracle BPEL Process Manager > BPEL Control.
2. Log in as oc4jadmin/password.

Testing BPEL Processes  20-17
where password is the oc4jadmin password you entered during installation.

3. Click Instances.

4. Select a specific instance from which to create test file content.

5. Click Test.

   If the instance has not yet completed or if the test was already initiated as a test case, you cannot save this instance as a test case.

6. Select a method:
   
a. Click Save as unit test (.xml) to save the contents to a test file. In this case, all partners are emulated.

   b. Click Save as composite test (.zip) to save the contents of multiple tests to a zip file. In this case, BPEL partners are called with test case information while other partners (like human workflow) are emulated. You must then unzip this file before importing the tests into Oracle JDeveloper. Note that with composite tests, tests can call other tests. If you change the name of a composite test file, ensure that you edit any test files that may call this file to include the correct name.

7. Save the file as XML to a directory location.

8. Return to Oracle JDeveloper.

9. If you saved the file as a composite test file, unzip it.

10. Right-click the Test Suites folder and select Import BPEL Test.

11. Select the test suite in the BPEL process in which to import the test.

12. Click the flashlight icon for the Imported Test URL field.

13. Select the file saved in Step 7 and click Open.

14. Click OK.

   The test is added to the selected test suite in the Application Navigator.

15. Right-click the test suite and select Edit Test Suite.

   Note that the test and its contents now display in the Edit Test Suite window.

   **Note:** No assertions are created in test cases generated from Oracle BPEL Control.
Deploying a Test Suite

After creating a test suite of test cases, you deploy the test to Oracle BPEL Server. Two deployment methods are provided:

- Deploying from Oracle JDeveloper
- Deploying from an ant Task

Deploying from Oracle JDeveloper

Oracle JDeveloper can be used to deploy test suites manually when you are first create them. Follow these steps to deploy test suites from Oracle JDeveloper.

1. Ensure that you first deploy the BPEL process with which the test suite is associated. The process and the test suite must be deployed separately and cannot be deployed together.
2. Right-click the Test Suites folder and select BPEL Test Deployer. The BPEL Test Deployer window appears.
3. Click the check box for the test suite to deploy.
4. Expand test_suite_name - Deploy > Tests and select the specific test cases to deploy.
5. Select the server to which to deploy the tests.
6. Click Deploy to compile, validate, and deploy the selected test cases of the test suite. Deployment status messages display in the Status section.
Deploying from an ant Task

An ant task can be used to deploy test suites. This task is useful in an automated testing environment.

**Note:** The BPEL process must be deployed before tests for a process can be deployed.

Follow these steps to deploy test suites from an ant task.

1. View the ant task parameters used to deploy test suites in the build.xml file.
   This file is located in the Resources folder of your BPEL process.

   ```xml
   <target name="test" depends="deployTestSuites, bpelTest, report" />
   <target name="prepareTests">
     <echo>
     | Preparing BPEL tests for deployment
     </echo>
     <delete file="${process.dir}/output/bpeltest.zip" quiet="true"/>
     <zip basedir="${process.dir}/bpel/testsuites" filesonly="true" 
       excludes="test_suites.xml, **/excludes/*" 
       destfile="${process.dir}/output/bpeltest.zip"/>
     </target>
   <target name="deployTestSuites" depends="prepareTests">
     <echo>
     | Deploying bpel tests ${process.name} on ${http.hostname}, port ${http.port}
     </echo>
     <deployTestSuites
       user="${admin.user}"        password="${admin.password}" 
       hostname="${http.hostname}" httpport="${http.port}" 
       domain="${domain}"          process="${process.name}" 
       rev="${rev}" 
       testfile="${process.dir}/output/bpeltest.zip"/>
   </target>
   </target>
   
   Note: The BPEL process must be deployed before tests for a process can be deployed.
   
   2. If you want to change the parameter values for this process, edit and uncomment appropriate parameters in the build.properties file located in the same Resources folder as the build.xml file. Note that this changes the values for this process only.

   ```properties
   #domain=default
   #rev=1.0
   #user=oc4jadmin
   #hostname=localhost
   #http.hostname=localhost
   #http.port=9700
   #j2ee.hostname=localhost
   #j2ee.port=23791
   #oc4jinstancename=home
   #asinstancename=
   #opmn.requestport=6000
   #platform=ias_10g
   #platform=oc4j_10g
   #bpeltest.callHandler=
   #bpel.test.context.properties=${bpel.home}/samples/tutorials/102.InvokingProcesses/rm
   ```
Running a Test Suite and Viewing Report Results

3. Open the developer’s prompt by selecting Start > All Programs > Oracle - Oracle_Home > Oracle BPEL Process Manager > Developer Prompt.

4. Change directories to the correct location and run ant to deploy the test suite:
   ```
   ant test
   ```
   where test is the target name defined in the build.xml file.

Running a Test Suite and Viewing Report Results

After deployment, you can run the test cases of a test suite on a BPEL process instance and view XML document reports. By default, report results are formatted as JUnit XML test results. Two methods are provided:

- Running from Oracle BPEL Control
- Running from an ant Task

**Note:** Test results are stored as binary large objects (BLOBs).

Running from Oracle BPEL Control

Oracle BPEL Control can be used to run tests manually and generate report results.

Follow these steps to run test suites and view report results from Oracle BPEL Control.

1. Log into Oracle BPEL Control by selecting Start > All Programs > Oracle - Oracle_Home > Oracle BPEL Process Manager > BPEL Control.

2. Log in as oc4jadmin/password
   where password is the oc4jadmin password.

3. Click BPEL Processes.

4. Click the instance to test.

5. Click Test Suites.

   If a test suite was deployed with this instance, the following window appears.

   If a test suite has not been deployed with this instance, this window does not appear. You must first deploy a test suite.

6. Select the check box for the test suite (for this example, named main).
Check boxes for the tests in the test suites appear.

7. Select the XML tests you want to run.

You can also undeploy test suites by clicking Undeploy Tests.

8. Click Execute Tests.

As tests complete, the window is updated with results in three sections:

- Test details such as process name, test run dates, which suites ran, and the number of workers appear. A worker is the number of concurrent threads that can run one or more test cases. The number of workers controls how many test cases can be run simultaneously.

- A summary report displays the outcomes of the tests and code coverage.

- A detailed listing of all the tests and which assertions failed is displayed. Note the Display Failures Only check box. When checked, detailed information for successful tests is hidden. By default, this check box is checked.
Running a Test Suite and Viewing Report Results

BPEL Test Report

<table>
<thead>
<tr>
<th>Test Parameters</th>
<th>PriceFinderWithTests (v. 1.0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start date</td>
<td>Saturday, June 24, 2006 3:10:52 PM</td>
</tr>
<tr>
<td>Completion Date</td>
<td>Saturday, June 24, 2006 3:10:54 PM</td>
</tr>
<tr>
<td>Test Suites</td>
<td>logicSuite</td>
</tr>
<tr>
<td>Workers</td>
<td>1</td>
</tr>
</tbody>
</table>

Summary Report

<table>
<thead>
<tr>
<th></th>
<th>Total</th>
<th>Initiated</th>
<th>Completed</th>
<th>Successful</th>
<th>Success Rate</th>
<th>Process Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0%</td>
<td>7.3%</td>
</tr>
</tbody>
</table>

Detail Report

- Display Failures Only
- Test: Shipping Considered (logicSuite) | Test-ID: ...

Activities that were executed are framed in green and those that were not are framed in red.

9. Click the percentage value link under Process Coverage to view the amount of source code executed.

10. Click one of the following links in the upper right corner of Oracle BPEL Control to view specific code coverage details.
Running a Test Suite and Viewing Report Results

11. Click View Instance Flow to view additional details or click Back to BPEL Control.

Running from an ant Task

An ant task can be used to run test suites and generate report results. This task is useful in an automated testing environment. By default, the test results are formatted as JUnit XML test results. This format provides the following benefits:

- Uses the junitreport task to produce a frames-based report.
- Integrates your BPEL test results with other JUnit results (if you use JUnit to test your Java components).
- Integrates with other third-party test suite frameworks that also use the JUnit report format.

Follow these steps to run tests and create reports from an ant task.

1. View the ant task parameters used to run test cases and generate reports in the build.xml file. This file is located in the Resources folder of your BPEL process.

```xml
<target name="bpelTest">
  <echo>
    EXECUTING process ${process.name}(v.${rev}):
    minCoverage=${bpeltest.minCoverage}, timeout=${bpeltest.timeout} sec,
    numWorkers=${bpeltest.numWorkers}
  </echo>
  <delete dir="${bpeltest.results.dir}/xml/${process.name}" quiet="true"/>
  <bpeltest
    user=${admin.user} password=${admin.password} hostname=${http.hostname} port=${http.port} domain=${domain} process=${process.name} timeout=${bpeltest.timeout} numWorkers=${bpeltest.numWorkers} minCoverage=${bpeltest.minCoverage}/>
</target>
```
Running a Test Suite and Viewing Report Results

```xml
<target name="report">
  <echo>
    Executed ${test.total.count} test(s) for ${process.name} (v.${rev}) with
    ${test.failure.count} failure(s)
  </echo>
</target>
```

2. If you want to change the parameter values for this process, see Step 2 on page 20-20 for details.

3. Open the developer’s prompt by selecting Start > All Programs > Oracle - Oracle_ Home > Oracle BPEL Process Manager > Developer Prompt.

4. Change directories to the correct location and start ant to run the test suite and create the reports:

```xml
<!-- Convenience targets --
<target name="deploy_test" depends="deploy, test"/>
</project>
```
Running a Test Suite and Viewing Report Results

ant bpelTest report

where bpelTest and report are the target names defined in the build.xml file.

Note: If you want to deploy the BPEL process, deploy the corresponding test cases, run the deployed test cases, and generate reports at the same time, run the following command:

ant deployProcess test

The following example shows an ant-generated report:

Since JUnit is being used, several Java constructs such as packages and classes are included in the report. For Oracle BPEL Process Manager, the default package resolves to bpel.domain.process-revision, but this can be customized. Classes map to test suite names. Note the code coverage suite. This is added when you specify a minimum code coverage in the ant task. In the main pane, a summary of the test results is shown. If you package results from multiple processes in this report, there are more entries in the Packages table.

5. Click the package name (BPEL process) to display report results.
Advanced Test Suite Design Features

This section describes several advanced test suite design features. Samples for some of these features are available in the `SOA_Oracle_Home\bpel\samples\references` subdirectories.

- Setting Dynamic Values at Run Time
- Asynchronous Event Emulation
- Verifying External Actions
- Custom Reporting
- Database Views
- XML Schemas
- Client APIs

Setting Dynamic Values at Run Time

You may sometimes want to have messages that contain dynamic content. For example:

- You want to emulate a message that contains the current date and time
- You need a unique identifier

To support this, you add message updates to emulated messages. A message update consists of two main pieces of information:

- An XPath function that generates a string value
- An XPath expression that specifies the portion of the message to update
For example, assume you want to emulate the following message:

```xml
<dateTimeMessage xmlns="http://xmlns.oracle.com/SomeProcess">
  <theDateTime>current date time</theDateTime>
</dateTimeMessage>
```

Add an emulation with the following content in Oracle JDeveloper:

```xml
<dateTimeMessage xmlns="http://xmlns.oracle.com/SomeProcess">
  <theDateTime/>
</dateTimeMessage>
```

Since there is no graphical user interface support for message updates, switch to the source view and view the following generated emulation:

```xml
<activityDriver name="someActivity">
  <emulate>
    <inboundMessage>
      <part name="payload">
        <content>
          <dateTimeMessage xmlns="http://xmlns.oracle.com/SomeProcess">
            <theDateTime/>
          </dateTimeMessage>
        </content>
      </part>
    </inboundMessage>
  </emulate>
</activityDriver>
```

Add a message update to the `inboundMessage` to set the date time at run time:

```xml
<activityDriver name="Invoke_2">
  <emulate duration="PT">
    <inboundMessage>
      <part name="payload">
        <content>
          <dateTimeMessage xmlns="http://xmlns.oracle.com/SomeProcess">
            <theDateTime/>
          </dateTimeMessage>
        </content>
        <update>
          <location>
            /client:dateTimeMessage/client:theDateTime
          </location>
          <XPathExpression>
            xp20:current-dateTime()
          </XPathExpression>
          <update/>
        </update>
      </part>
    </inboundMessage>
  </emulate>
</activityDriver>
```

Note that the `update` element contains an XPath location to the empty `theDateTime` element. The XPath expression element contains an XPath function that generates the current date time. When the test is run, this XPath function is called and the emulated message is updated with the current date time.

**Note:** Namespace prefixes used in the XPath function and XPath query must be correctly defined in the test case XML document.
Asynchronous Event Emulation

It is possible to emulate asynchronous events. These events are captured in the onMessage or onAlarm branches in your BPEL process. There is no current design time support for event emulation.

Verifying External Actions

You can extend your test cases beyond BPEL. For example, assume you have a BPEL process that alters an external system. You can write a test case that invokes an instance of the BPEL process and calls an API that verifies that the external system was updated appropriately. Since there is limited support for this in the BPEL test framework, you can write custom XPath functions to verify data and then use these functions in assertions. There is no design time support for this. To do this manually, write an assertion as follows:

```xml
<activityDriver name="Assign_1">
  <assertValue fatal="true">
    <message>exact text comparison</message>
    <actualPath>ns:extractExternalValue()</actualPath>
    <expected>theInput</expected>
  </assertValue>
</activityDriver>
```

In the bold section, note that an XPath function was used instead of the usual XPath query. This function can be implemented to extract a value from an external system The BPEL test framework then compares the value returned by the XPath function to theInput.

Custom Reporting

The ant task generates JUnit-style XML results by default. These results can then be passed to the junitreport task to generate a frames-based HTML report. If you want to present the results in a different format, set the xsl parameter of the bpelTest ant task to a URL or file path that points to an XSLT stylesheet you created. After tests have been executed, an XSL transformation is performed on the results XML document (specified by the XML schema defined in SOA_Oracle_Home/bpel/system/xmllib/BPELTestResult.xsd) using the stylesheet specified by the xsl attribute.

The following example shows the ant task using an XSL stylesheet accessible from a URL:

```xml
<bpelTest ... xsl="http://mycompany.com/myStyleSheet.xsl"/>
```
The following example shows the ant task using an XSL stylesheet accessible from the file system:

```xml
<bpelTest ... xsl="C:\xslDir\myStyleSheet.xsd" .../>
```

**Database Views**

The following database views enable you to query details about test results and test definitions.

- `admin_list_td`
- `admin_list_tdef`

**admin_list_td**

This view provides information about previously run test cases, as shown in Table 20-1.

```
SQL> select cikey, ci_domain_ref, test_run_name from admin_list_td;
CIKEY | CI_DOMAIN_REF | TEST_RUN_NAME
-------+----------------+--------------------------------------------
   4 |             0 | ca8f3d34e7fa93f2:53855ec5:10c02daf400:-7b46

SQL> select test_run_id from admin_list_td;
TEST_RUN_ID
-------------------------------------------
ca8f3d34e7fa93f2:53855ec5:10c02daf400:-7b46
```

Table 20–1  `admin_list_td`

<table>
<thead>
<tr>
<th>Name</th>
<th>Null</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIKEY</td>
<td>N</td>
<td>VARCHAR2(100)</td>
<td>The BPEL process</td>
</tr>
<tr>
<td>CI_DOMAIN_REF</td>
<td>N</td>
<td>SMALLINT(5)</td>
<td>The BPEL process domain</td>
</tr>
<tr>
<td>TEST_RUN_NAME</td>
<td>N</td>
<td>VARCHAR2(100)</td>
<td>The name of the test run. This can stay constant across many test runs. This value is automatically generated unless the test run was initiated through the client code.</td>
</tr>
<tr>
<td>TEST_RUN_ID</td>
<td>N</td>
<td>VARCHAR2(100)</td>
<td>The ID of the test run. This must be unique across test runs. This value is automatically generated unless the test run was initiated through the client code.</td>
</tr>
<tr>
<td>TEST_SUITE</td>
<td>N</td>
<td>VARCHAR2(100)</td>
<td>The test suite of this test case</td>
</tr>
<tr>
<td>TEST_LOCATION</td>
<td>N</td>
<td>VARCHAR2(100)</td>
<td>The location of the test case within the test suite. This is typically the name of the test case file.</td>
</tr>
<tr>
<td>TEST_STATUS</td>
<td>N</td>
<td>VARCHAR2(50)</td>
<td>The status of this test case. This is either passed, failed, or running.</td>
</tr>
<tr>
<td>TEST_RESULT</td>
<td>N</td>
<td>BLOB</td>
<td>The XML result of the test case. This column can be read using a DOM parser.</td>
</tr>
</tbody>
</table>

For example:

```
SQL> select cikey, ci_domain_ref, test_run_name from admin_list_td;
```

```
CIKEY | CI_DOMAIN_REF | TEST_RUN_NAME
-------+----------------+--------------------------------------------
   4 |             0 | ca8f3d34e7fa93f2:53855ec5:10c02daf400:-7b46
```

```
SQL> select test_run_id from admin_list_td;
```

```
TEST_RUN_ID
-------------------------------------------
ca8f3d34e7fa93f2:53855ec5:10c02daf400:-7b46
```

20-30  Oracle BPEL Process Manager Developer's Guide
Advanced Test Suite Design Features

SQL> select test_location, test_suite, test_status, test_result from admin_list_td;

<table>
<thead>
<tr>
<th>TEST_LOCATION</th>
<th>TEST_SUITE</th>
<th>TEST_STATUS</th>
<th>TEST_RESULT</th>
</tr>
</thead>
<tbody>
<tr>
<td>testShippingConsidered.xml</td>
<td>logicSuite</td>
<td>failed</td>
<td>&lt;?xml version...</td>
</tr>
</tbody>
</table>

admin_list_td

This view provides information about deployed test cases, as shown in Table 20-2.

Table 20-2 admin_list_td

<table>
<thead>
<tr>
<th>Name</th>
<th>Null</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROCESS_ID</td>
<td>N</td>
<td>VARCHAR2(100)</td>
<td>The BPEL process</td>
</tr>
<tr>
<td>REVISION_TAG</td>
<td>N</td>
<td>VARCHAR2(50)</td>
<td>The BPEL process revision</td>
</tr>
<tr>
<td>DOMAIN_REF</td>
<td>N</td>
<td>SMALLINT(5)</td>
<td>The BPEL process domain</td>
</tr>
<tr>
<td>TEST_SUITE</td>
<td>N</td>
<td>VARCHAR2(100)</td>
<td>The test suite of this test case. This is typically the name of the test case file.</td>
</tr>
<tr>
<td>LOCATION</td>
<td>N</td>
<td>VARCHAR2(100)</td>
<td>The location of the test case within the test suite. This is typically the name of the test case file.</td>
</tr>
<tr>
<td>TYPE</td>
<td>N</td>
<td>VARCHAR2(10)</td>
<td>The type of this file:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- message — a message file</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- include — a baseline test case</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- test — a test case</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- properties — a properties file</td>
</tr>
<tr>
<td>CREATION_DATE</td>
<td>N</td>
<td>DATE</td>
<td>The date this test was deployed</td>
</tr>
<tr>
<td>DEFINITION</td>
<td>N</td>
<td>BLOB</td>
<td>The XML definition of the test case. This column can be read using a DOM parser.</td>
</tr>
</tbody>
</table>

SQL> select process_id, revision_tags, domain_ref, test_suite from admin_list_td;

<table>
<thead>
<tr>
<th>PROCESS_ID</th>
<th>REVISION_TAG</th>
<th>DOMAIN_REF</th>
<th>TEST_SUITE</th>
</tr>
</thead>
<tbody>
<tr>
<td>PriceFinderWithTests</td>
<td>1.0</td>
<td>0</td>
<td>logicSuite</td>
</tr>
<tr>
<td>PriceFinderWithTests</td>
<td>1.0</td>
<td>0</td>
<td>logicSuite</td>
</tr>
<tr>
<td>Emulations</td>
<td>1.0</td>
<td>0</td>
<td>main</td>
</tr>
<tr>
<td>Emulations</td>
<td>1.0</td>
<td>0</td>
<td>main</td>
</tr>
<tr>
<td>Emulations</td>
<td>1.0</td>
<td>0</td>
<td>main</td>
</tr>
<tr>
<td>Emulations</td>
<td>1.0</td>
<td>0</td>
<td>main</td>
</tr>
<tr>
<td>Emulations</td>
<td>1.0</td>
<td>0</td>
<td>main</td>
</tr>
<tr>
<td>Emulations</td>
<td>1.0</td>
<td>0</td>
<td>main</td>
</tr>
<tr>
<td>Emulations</td>
<td>1.0</td>
<td>0</td>
<td>main</td>
</tr>
<tr>
<td>Emulations</td>
<td>1.0</td>
<td>0</td>
<td>main</td>
</tr>
</tbody>
</table>

SQL> select location, type, creation_date, definition from admin_list_td;

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>TYPE</th>
<th>CREATION_DATE</th>
<th>DEFINITION</th>
</tr>
</thead>
<tbody>
<tr>
<td>baseline.xml</td>
<td>include</td>
<td>2006-06-25 16:58:16.260000</td>
<td>&lt;?xml version...</td>
</tr>
</tbody>
</table>
Advanced Test Suite Design Features

XML Schemas

The following XML schemas are provided in the SOA_Oracle_Home\bpel\system\xmllib directory:

- InstanceDriver.xsd — Test case schema
- BPELTestResult.xsd — Test results schema

Client APIs

A service defined in the com.oracle.services.bpel.test package enables you to use the Locator class to look up the test service and initiate test cases, get a list of deployed tests, or get the results of test cases already executed. This service can be looked up in a similar manner to the delivery service. For the service name, you can use com.oracle.services.bpel.test.ITestService.SERVICE_NAME.

See Also:
- Oracle BPEL Process Manager Client API Reference:
  SOA_Oracle_Home\bpel\docs\apidocs
Oracle BPEL Portlets consist of Oracle BPEL Control report portlets and Oracle BPEL Worklist Application portlets. This chapter describes how to deploy the Oracle BPEL Portlets and configure the Oracle Application Server Portal (OracleAS Portal) to provide access to data from these portlets.

This chapter contains the following topics:

- OracleAS Portal Introduction
- Step 1: Installing and Configuring the Required Oracle Application Server Components
- Step 2: Deploying the Portlets
- Step 3: Registering Web Providers with OracleAS Portal
- Step 4: Defining Portlet Parameters and Accessing Portlet Data Sources
- Step 5: Mapping Portlet Parameters with Page Parameters
- Summary

**OracleAS Portal Introduction**

OracleAS Portal is a component of Oracle Application Server used for the development, deployment, administration, and configuration of enterprise class portals. OracleAS Portal enables you to present information from multiple, unrelated data sources in a single organized view. This view, a portal page, can contain one or more components called portlets that can each get their content from different data sources.

A portlet parameter provides a configurable way of passing a value to a portlet. Using portlet parameters, the information displayed in a portlet can be specific to a particular page or user. Portlet parameters are created by the portlet developer and exposed to the page designer through the user interface. After adding a portlet to a page, page designers can assign values to the public portlet parameters to make the information displayed in the portlet specific to the page.

Web providers enable you to deploy portlets to OracleAS Portal. They may reside on the same application server as OracleAS Portal, on a remote application server, or anywhere on the network. Web providers use open standards, such as XML, SOAP, HTTP, or J2EE for deployment, definition, and communication with OracleAS Portal. 

Figure 21–1 provides an overview of this integration.
You can configure OracleAS Portal to access the following portlets:

- **Oracle BPEL Control reports portlets**
  Enables you to access instance state, instance execution time, performance, activity sensor, fault sensor, and process time distribution reports that describe the behavior of instances of a selected BPEL process. Oracle BPEL Control reports enable you to:
  - Receive an overall view of business process instance performance
  - Analyze data for the BPEL process instances and make critical decisions
  - Analyze data of the activities that constitute a business process
  - Identify and debug faults and take appropriate corrective actions

  **See Also:** The following documentation for additional descriptions of these report types:
  - "Defining Oracle BPEL Control Report Portlet Parameters and Accessing Portlet Data Sources" on page 21-9
  - "Creating Oracle BPEL Control Reports" on page 22-1

- **Oracle BPEL Worklist Application portlets**
  Enables you to access Oracle BPEL Worklist Application, a Web interface that enables users to access and act on tasks assigned to them. The tasks displayed depend on the user’s profile, and the actions allowed depend on the user’s privileges. The Oracle BPEL Worklist Application is layered on top of the BPEL worklist service.

  **See Also:** The following documentation for descriptions of Oracle BPEL Worklist Application portlets and for additional details about Oracle BPEL Worklist Application:
  - "Defining Oracle BPEL Worklist Application Portlet Parameters and Accessing Portlet Data Sources" on page 21-13
  - Chapter 16, "Worklist Application"
Step 1: Installing and Configuring the Required Oracle Application Server Components

Oracle Application Server (OracleAS) Portal uses Oracle Internet Directory, an LDAP-compliant directory that provides a single repository and administration environment for user accounts. Oracle BPEL Worklist Application also requires the use of Oracle Internet Directory; Oracle BPEL Control does not require this component. Therefore, to use both of these Oracle BPEL Portlets, you must install the following installation types to use OracleAS Portal:

- Install the Identity Management and Metadata Repository installation type of Oracle Application Server Infrastructure 10g Release 2 (10.1.2.0.2).
- Install Portal and Wireless middle tier 10g Release 2 (10.1.2.0.2).
- Install BPEL Process Manager for OracleAS Middle Tier 10.1.3.1.0.

**Notes:**

- OracleAS Portal cannot be configured for use with the BPEL Process Manager for Developers installation type. This is because BPEL Process Manager for Developers does not use Oracle Internet Directory (required by OracleAS Portal).
- You must use the 10.1.2.0.2 Portal and Wireless middle tier install type with Oracle BPEL Process Manager 10.1.3.1.0. This middle tier version enables you to view Oracle BPEL Worklist Application portlets from both 10.1.2.0.2 and 10.1.3.1.0.

**See Also:** Oracle BPEL Process Manager Administrator’s Guide for details about the demo user community

### Configuring Realms (10.1.3.1.0 Only)

Multiple realms are supported in 10.1.3.1.0 through customization of the `SOA_Oracle_Home\bpel\system\services\config\is_config.xml` file. The `is_config.xml` file must contain a configuration (realm) that corresponds to the Oracle Internet Directory instance of the portal where the portlets are to be used. Since `is_config.xml` can be configured to support multiple realms, the `wf_client_config.xml` file must specify the realm name from `is_config.xml` that corresponds to the portal Oracle Internet Directory instance.

1. Use a text editor to open the `wf_client_config.xml` file located in the same directory as `is_config.xml`.

**See Also:** The service configuration chapter of Oracle BPEL Process Manager Administrator’s Guide for instructions on configuring Oracle Internet Directory 10.1.2.0.2 with Oracle BPEL Process Manager 10.1.3.1.0

**Note:** If you install the identity service demo user community and run the workflow service samples shipped with Oracle BPEL Process Manager, you can display data about these users in the Oracle BPEL Worklist Application portlets.

**See Also:** Oracle BPEL Process Manager Administrator’s Guide for details about the demo user community
2. Change jazn.com to the realm value you specified in is_config.xml. This value is used for authenticating users coming from OracleAS Portal. For example, if the configuration name for the portal Oracle Internet Directory instance is specified by realm name portalOID in is_config.xml, then the portal realm mapping information in wf_client_config.xml is as follows:

```xml
<portal>
  <realmMapping>
    portalOID
  </realmMapping>
</portal>
```

See Also: “Multirealm Support” on page 15-107

### Step 2: Deploying the Portlets

You must install Oracle BPEL Process Manager and deploy the Oracle BPEL Portlets .ear file. Use one of the following methods to deploy the .ear file:

- Deploying Portlets with dcmctl
- Deploying Portlets with Oracle Enterprise Manager 10g Application Server Control Console

#### Deploying Portlets with dcmctl

Follow these procedures to deploy the portlets with the dcmctl utility.

1. Change directories to the following location:

   ```bash
   SOA_Oracle_Home\dcm\bin
   ```

2. Enter the following command to deploy Oracle BPEL Portlets:

   ```bash
   dcmctl deployApplication -f SOA_Oracle_Home\bpel\system\services\lib\BPELPortlet.ear -a BPELPortlet -co OC4J_BPEL -pa orabpel
   ```

   When deployment completes successfully, a message similar to the following appears:

   ```bash
   Application:    bpelportlet
   Component Name: OC4J_BPEL
   Component Type: OC4J
   Instance:       mark.my-pc.us.oracle.com
   ```

   See Also: Oracle Application Server Administrator’s Guide for additional information about the dcmctl utility

#### Deploying Portlets with Oracle Enterprise Manager 10g Application Server Control Console

Follow these procedures to deploy the portlets with Oracle Enterprise Manager 10g Application Server Control Console.

1. Go to the following URL:

   ```bash
   http://hostname:port/em
   ```

   where:

21-4 Oracle BPEL Process Manager Developer’s Guide
Step 2: Deploying the Portlets

Oracle BPEL Portlets

21-5

hostname is the name of the host on which Oracle BPEL Process Manager is installed.

port is the number of the Oracle HTTP Server port. The following sources also identify the port number used to access the Oracle Enterprise Manager 10g Application Server Control Console:

- The bpelsetupinfo.txt file in the install subdirectory of the Oracle home for your installation
- The SOA_Oracle_Home\bpel\utilities\ant-orabpel.properties file.
- The following command:

  prompt> SOA_Oracle_Home\open\bin\opmnctl status -1

Note: On Windows platforms, you can also start Oracle Enterprise Manager 10g Application Server Control Console from the Start Menu.

2. Enter the following login details when prompted:

   Field | Value
   ---   | ---
   Username | Enter oc4jadmin
   Password | Enter the password you specified during installation for the oc4jadmin user.

   The Oracle Application Server home page appears.

3. Select the OC4J instance in which Oracle BPEL Process Manager is deployed.

4. Click the Applications tab.

5. Click Deploy EAR file.
   The Deploy Application window appears.

6. Enter the following details:

   Field | Description
   ---   | ---
   Archive Location | Select the SOA_Oracle_Home\bpel\system\services\lib\BPELPortlet.ear file.
   Application Name | Enter BPELPortlet
   Parent Application | Select orabpel
   Context Root | Enter BPELPortlet

7. Click Next.

8. Enter the following details:

   Field | Description
   ---   | ---
   Application Name | Enter BPELPortlet
   Parent Application | Select orabpel
   Context Root | Enter BPELPortlet

Oracle BPEL Portlets
9. Click Next.
   The Deployment Settings window appears.

10. Go to the task Select security provider.

11. Select Oracle Identity Management from the drop-down list.
   The Oracle Internet Directory host and port information automatically appears if
   the orabpel parent application is configured with Oracle Internet Directory as the
   security provider.

12. Click Deploy.
   Messages display indicating that deployment is in progress. When complete, the
   following message appears:

   Application "BPELPorlet" was successfully deployed.

Step 3: Registering Web Providers with OracleAS Portal

Web providers provide portlet data to OracleAS Portal. You must first register the
Oracle BPEL Process Manager Web provider with the specific OracleAS Portal
instance. This enables you to use the Oracle BPEL Control report portlets and Oracle
BPEL Worklist Application portlets in OracleAS Portal pages.

Once you have registered these Web providers, you can access the data provided by
the portlets.

1. Go to the 10.1.2.0.2 OracleAS Portal home page. For example:

   http://hostname:port_number/pls/portal

   where hostname is the host on which the 10.1.2.0.2 Portal and Wireless middle
tier is installed and port_number is typically 7777. If you are unsure, see the
setupinfo.txt file in the SOA_Oracle_Home\install directory for your
Portal and Wireless installation. You can also access OracleAS Portal from a link on
the Oracle Application Server Welcome page.

2. Log in to the OracleAS Portal instance using the portal user name and ias_admin
   password. This is the same password that you specified during Portal and
   Wireless middle tier installation and BPEL Process Manager for OracleAS Middle
   Tier installation.

3. Select the Administrator tab, then the Portlets subtab to access the Portal Builder
   window to register the provider.

4. Click Register a Provider in the Remote Providers section.

5. Enter the following details on the Provider Information page:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Enter a unique and recognizable name.</td>
</tr>
<tr>
<td>Display Name</td>
<td>Enter a name to display on the portlet page header.</td>
</tr>
<tr>
<td>Timeout</td>
<td>Enter the amount of time a page takes to render if the portlet is not responding.</td>
</tr>
<tr>
<td>Timeout Message</td>
<td>Enter message to display when a timeout occurs.</td>
</tr>
<tr>
<td>Implementation Style</td>
<td>Ensure that Web is selected.</td>
</tr>
</tbody>
</table>

21-6 Oracle BPEL Process Manager Developer's Guide
6. Click Next.
7. Enter appropriate details on the General Properties page. For the URL of the Web provider, ensure that you enter the following details for Oracle BPEL Process Manager:
   
   http://bpel_host:bpel_port/BPELPortlet/providers
   
   where $bpel_host$ is the host on which Oracle BPEL Process Manager is installed and $bpel_port$ is the port. For BPEL Process Manager for OracleAS Middle Tier installations, the port is typically 7777 or 7778. If you are unsure of the port, see the $SOA_Oracle_Home\install\bpelsetupinfo.txt$ file that was created after BPEL Process Manager for OracleAS Middle Tier installation.
8. Select Once Per User Session from the Login Frequency list.
9. Click Next.
10. Enter your user or group name in the Grantee field of the Control Access page, click Add, and then click Finish. If you are not sure about the user or group name, click the appropriate browse icon.
    
    If provider registration was successful, you receive a success message.
11. Click OK.
12. Click Builder, then Build to display the Portal Builder page.
13. Go to the Page Groups section of the Portal Builder page.
14. Select the group from the Page Group list. If a page group does not exist, create one.
15. Click Create a Page and follow the steps that appear.
    
    The new page appears with the display name header you entered when creating the page.
16. Click the + icon (second icon from the left) to add an instance of the portlet to the page.

The Add Portlets To Region page appears.
17. Click Portlet Staging Area in the Portlet Repository section.

18. Click the portlet you previously created in the Available Portlets section.

This displays the Oracle BPEL Portlets in the Available Portlets section.

Note: The following portlet link titles in the Available Portlets section apply to these report types:

- The portlet link entitled This report shows the process performance data based on completion time SLA applies to performance reports.
- The portlet link entitled This report shows the distribution of the process execution time across all the activities applies to process time distribution reports.

You can always change these titles when editing the portlet.
19. Click the portlets you want.
   The selected portlets are added to the Selected Portlets section.
20. Click OK.
21. See "Step 4: Defining Portlet Parameters and Accessing Portlet Data Sources" on page 21-9 for information on setting portlet properties.

   See Also: See Oracle Application Server Portal User's Guide for additional details about using OracleAS Portal

Step 4: Defining Portlet Parameters and Accessing Portlet Data Sources

When you access a portlet for the first time, some default settings already appear for Oracle BPEL Worklist Application. No default settings appear for Oracle BPEL Control reports. For example, if you selected Instance Execution Time in Step 19 on page 21-9, the page appears with the following message:

1. Click the Edit Defaults icon (first icon on the left above the message) to set the parameter values of the Oracle BPEL Control reports portlets or change the default settings for Oracle BPEL Worklist Application portlets.
2. See the following section based on your portlet selection in Step 1.

<table>
<thead>
<tr>
<th>If You Selected...</th>
<th>Go to Section...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instance State</td>
<td>&quot;Defining Oracle BPEL Control Report Portlet Parameters and Accessing Portlet Data Sources&quot; on page 21-9</td>
</tr>
<tr>
<td>Instance Execution Time</td>
<td>&quot;Defining Oracle BPEL Control Report Portlet Parameters and Accessing Portlet Data Sources&quot; on page 21-9</td>
</tr>
<tr>
<td>Performance</td>
<td></td>
</tr>
<tr>
<td>Activity Sensor</td>
<td></td>
</tr>
<tr>
<td>Process Time Distribution</td>
<td></td>
</tr>
<tr>
<td>Fault Sensor</td>
<td></td>
</tr>
<tr>
<td>BPEL Worklist Portlet</td>
<td>&quot;Defining Oracle BPEL Worklist Application Portlet Parameters and Accessing Portlet Data Sources&quot; on page 21-13</td>
</tr>
</tbody>
</table>

Defining Oracle BPEL Control Report Portlet Parameters and Accessing Portlet Data Sources

Report portlets provide access to Oracle BPEL Control reports that describe the behavior of the instances of a selected BPEL process.
Each of the report portlets must be associated with a specific time period. Table 21–1 describes the parameter values to specify for each report type. This window provides a method for specifying the same parameter values across all report type portlets in a given page.

1. Enter the following parameter values.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title</td>
<td>Enter a report title name or accept the default name. The title that appears by default is based upon the type of report you selected in Step 1 on page 21-9.</td>
</tr>
<tr>
<td>Domain</td>
<td>Select the domain in which the BPEL process is deployed.</td>
</tr>
<tr>
<td>Business Process</td>
<td>Select the BPEL process for which to create reports of its instances.</td>
</tr>
<tr>
<td>Time Interval</td>
<td>Select Daily, Hourly, or Weekly as the time interval for generating data.</td>
</tr>
<tr>
<td>End Date</td>
<td>Enter the date at which to stop generating report data.</td>
</tr>
<tr>
<td>End Hour</td>
<td>Enter the time at which to stop generating report data.</td>
</tr>
<tr>
<td>Number of Time Intervals</td>
<td>Enter the time period for creating reports. This is the number of days, weeks, or hours (according to the Time Interval list selection) starting from the end date going backwards in time to determine the start date.</td>
</tr>
</tbody>
</table>

For example, if you enter 7/29/05 as the end date, 22:00 as the end hour, Weekly as the time interval, and 3 as the number of time intervals, the report is created for the three weeks between 7/08/05 at 10 PM and 7/29/05 at 10 PM.

2. Click Apply to apply your values and click OK to create portlet data.

3. See the following sections based on the portlet type you selected:

<table>
<thead>
<tr>
<th>Portlet</th>
<th>See Section...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instance State</td>
<td>&quot;Instance State&quot; on page 21-11</td>
</tr>
<tr>
<td>Instance Execution Time</td>
<td>&quot;Instance Execution Time&quot; on page 21-11</td>
</tr>
<tr>
<td>Performance</td>
<td>&quot;Performance&quot; on page 21-12</td>
</tr>
<tr>
<td>Activity Sensor</td>
<td>&quot;Activity Sensor&quot; on page 21-13</td>
</tr>
<tr>
<td>Fault Sensor</td>
<td>&quot;Fault Sensor&quot; on page 21-13</td>
</tr>
</tbody>
</table>
Step 4: Defining Portlet Parameters and Accessing Portlet Data Sources

Instance State
The instance state report portlet shown in Figure 21–2 provides details about the number of faulted, completed, and currently running instances of the BPEL process for the selected time period.

Figure 21–2  Instance State Report

Note: You can edit parameter settings by clicking the Edit icon (first icon on the left above the portlet title bar).

Instance Execution Time
The instance execution time report portlet shown in Figure 21–3 provides details about the maximum, minimum, and average execution times aggregated across all closed instances of the business process during the selected time period.
Figure 21–3  Instance Execution Time Report

Performance

The performance report portlet shown in Figure 21–4 provides aggregated information about the percentage of BPEL process instances that meet the service level agreement (SLA) value associated with a specific BPEL process.

Figure 21–4 Performance Report

See Also:  “Creating Performance Reports” on page 22-5 for a description of the SLA parameter and how to set it.
Activity Sensor
The activity sensor report portal shown in Figure 21–5 provides details about the activity sensors associated with the BPEL process over a selected time period. This enables you to analyze activity trends. Activity sensor reports show the activity name, sensor name, activity type, aggregated values of the minimum, maximum, and average execution time of the activity, and the number of occurrences of the activity. This report shows activity information only if activity sensors are defined for the BPEL process.

![Activity Sensor Portlet](image1)

<table>
<thead>
<tr>
<th>Activity Sensor</th>
<th>Parameter</th>
<th>Process</th>
<th>Start Date</th>
<th>End Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>processingTime</td>
<td>ErrorSensor</td>
<td>assign</td>
<td>2005-06-30 00:00:00</td>
<td>2005-06-30 23:59:59</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2005-06-30 00:00:00</td>
<td>2005-06-30 23:59:59</td>
</tr>
</tbody>
</table>

Process Time Distribution
Process time distribution reports enable you to view the distribution of execution times across the activities of BPEL process instances. For example, you may have two activities, one for each loan application company returning a loan offer. You can see which activity is taking the longest amount of time to process the loan offer. You do not need to create sensors in activities to use this report. This report enables you to identify which activities are taking the most time to process.

Fault Sensor
The fault sensor report portal shown in Figure 21–6 provides details about the data collected by fault sensors associated with a specific BPEL process over the specified time period. This enables you to analyze trends in faults. The report shows the date and time of the fault occurrence, the fault message, the activity in which the fault occurred, and a link to the faulted instance. This report shows activity information only if fault sensors are defined for the BPEL process. For this example, no fault sensor data was found.

![Fault Sensor Reports](image2)

<table>
<thead>
<tr>
<th>Fault Sensor</th>
<th>Parameter</th>
<th>Process</th>
<th>Start Date</th>
<th>End Date</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>faultSensor</td>
<td>2005-06-30 00:00:00</td>
<td>2005-06-30 23:59:59</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2005-06-30 00:00:00</td>
<td>2005-06-30 23:59:59</td>
</tr>
<tr>
<td></td>
<td>No Data Found</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Defining Oracle BPEL Worklist Application Portlet Parameters and Accessing Portlet Data Sources
These portlets integrate Oracle BPEL Worklist Application with OracleAS Portal. There are two types of portlets:
Listing portlet
Analysis portlet

Details about the parameter values to specify for these portlets are provided below.

See Also: Chapter 16, "Worklist Application" for additional details about Oracle BPEL Worklist Application

Listing Portlet Customization
The listing portlet displays a list of tasks that satisfy the data filtering criterion. This portlet provides considerable flexibility in terms of which tasks to display and how to display them.

The listing portlet exposes two portlet parameters: Task Category and Task Status. The values for these can be set from the page. If no value is set, then the default values of My & Group and Any are used. Figure 21–8 on page 21-15 shows the Page Parameter Properties page where these default values are set. See “Step 5: Mapping Portlet Parameters with Page Parameters” on page 21-20 for instructions on editing this page.
Step 4: Defining Portlet Parameters and Accessing Portlet Data Sources

Figure 21–8  Page Parameter Properties

<table>
<thead>
<tr>
<th>Parameter Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task Filter Customization</td>
</tr>
<tr>
<td>Category</td>
</tr>
<tr>
<td>Select one of the following:</td>
</tr>
<tr>
<td>My &amp; Group—tasks assigned to the user and the groups to which the logged-in user belongs</td>
</tr>
<tr>
<td>Reporters—tasks assigned to the users who report to the logged-in user</td>
</tr>
<tr>
<td>Owner—tasks that are owned by the logged-in user by way of process ownership</td>
</tr>
<tr>
<td>Creator—tasks that were created or initiated by the logged-in user</td>
</tr>
<tr>
<td>Previous—tasks that the logged-in user has updated</td>
</tr>
<tr>
<td>Status</td>
</tr>
<tr>
<td>Select the status state: Assigned, Completed, Errored, Expired, Withdrawn, Suspended, and Any</td>
</tr>
<tr>
<td>Created</td>
</tr>
<tr>
<td>Select the previous x days: (leave blank (that is, no restriction), 1 day, 7 days, 14 Days, and 30 days)</td>
</tr>
<tr>
<td>Expiration</td>
</tr>
<tr>
<td>Select the next x days: (leave blank (that is, no restriction), 1 day, 7 days, 14 days, and 30 days)</td>
</tr>
<tr>
<td>Priority</td>
</tr>
<tr>
<td>Select from Any or 1 through 5, where 1 is the highest priority</td>
</tr>
</tbody>
</table>

Oracle BPEL Portlets 21-15
2. Click Apply to apply your values and click OK to create portlet data.

Listing Portlet View

Figure 21–9 shows the task listing portlet for 10.1.2.0.2 and Figure 21–10 shows the task listing portlet for 10.1.3. The category of tasks and the current user name displays at the top. The table displays the columns chosen in the Display Customization options described in Table 21–2. The actual set of tasks that display depends on the Task Filter Customization options chosen in Table 21–2. The title column contains a link for clicking down into the worklist application for the task details. Paging information and the last page refreshed time are displayed at the bottom.

Table 21–2 (Cont.) BPEL Worklist Portlet Task Filter and Display Customization Options

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Business Process</td>
<td>Select one of the deployed business processes or Any</td>
</tr>
<tr>
<td>Display Customization</td>
<td></td>
</tr>
<tr>
<td>Title</td>
<td>Select the portlet title.</td>
</tr>
<tr>
<td>Layout</td>
<td>Select the size of the portlet as a ratio of the maximum size (Small, Medium, Large, or Full).</td>
</tr>
<tr>
<td>Show Header/Footer</td>
<td>Set this flag to Yes (default option) if the header (category and user name) and footer (page generation time) are to be displayed.</td>
</tr>
<tr>
<td>Column Layout (Columns 1 through 8)</td>
<td>Select the columns to display and the order in which to display them. There can be up to eight columns: Title, Number, Assignee, Status, Priority, Expiration, Last Modified, Created, Conclusion, Last Modifier, Creator, Acquirer, Task Key, and Business Process. Note that the first column is always Title.</td>
</tr>
<tr>
<td>Sort Column</td>
<td>Select the order in which to sort and display the following columns: Title, Number, Priority, Expiration, Conclusion, Last Modified, and Last Modifier.</td>
</tr>
<tr>
<td>Sort Order</td>
<td>Select Ascending or Descending.</td>
</tr>
<tr>
<td>Page Size</td>
<td>Select 5, 10, 20, or 50</td>
</tr>
<tr>
<td>Locale Source</td>
<td>Select the locale for the portlet based on the specified source: Portlet (resolved from the browser and set language portlet) or Directory Service.</td>
</tr>
<tr>
<td>Style Source</td>
<td>Select the style (background color, font color, and font style) for the portlet based on the specified source: Portlet (based on page style used in the portal page) or Default (Oracle BPEL Control or Oracle BPEL Worklist Application style).</td>
</tr>
</tbody>
</table>

Figure 21–9 Worklist Portlet View (10.1.2.0.2)

<table>
<thead>
<tr>
<th>Title</th>
<th>Number</th>
<th>Assignee</th>
<th>Status</th>
<th>Priority</th>
<th>Expiration</th>
<th>Modified</th>
<th>Conclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 21–10 Worklist Portlet View (10.1.3.0.0)

<table>
<thead>
<tr>
<th>Title</th>
<th>Number</th>
<th>Assignee</th>
<th>Status</th>
<th>Priority</th>
<th>Expiration</th>
<th>Modified</th>
<th>Conclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

21-16 Oracle BPEL Process Manager Developer's Guide
Step 4: Defining Portlet Parameters and Accessing Portlet Data Sources

Figure 21–10  Worklist Portlet View (10.1.3)

![Figure 21–10 Worklist Portlet View](image)

Figure 21–11 shows the 10.1.2.0.2 worklist application that you access by clicking a task entry under the Title column in Figure 21–9 on page 21-16. The worklist application recognizes that the user is clicking down from the portlet and sets the header information and home link accordingly. The search option is skipped and the home link points back to the portal page. All other behavior is the same as if the user had logged into the worklist application. If you click a 10.1.3 task entry under the Title column in Figure 21–10 on page 21-17, a similar worklist application appears.

Figure 21–11  Accessing the Worklist Application

![Figure 21–11 Accessing the Worklist Application](image)

Analysis Portlet Customization

Figure 21–12 shows the analysis listing portlet. The analysis portlet provides a status breakdown chart based on the category of tasks chosen and other filter options. It also provides the ability to group the results based on the assignee, business process, or the creator of tasks. The Listing Portlet exposes a portlet parameter for Task Category. The value for this can be set from the page. If no value is set, then the default value of My...
& Group is used. Figure 21–8 on page 21-15 shows the Page Parameter Properties page where this default value is set. See "Step 5: Mapping Portlet Parameters with Page Parameters" on page 21-20 for instructions on editing this page.

Figure 21–12  BPEL Worklist Analysis Portlet Filter and Display Customization Options

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task Filter Customization</td>
<td></td>
</tr>
<tr>
<td>Category</td>
<td></td>
</tr>
<tr>
<td>Created</td>
<td></td>
</tr>
<tr>
<td>Process</td>
<td></td>
</tr>
<tr>
<td>Business Process</td>
<td></td>
</tr>
<tr>
<td>Display Customization</td>
<td></td>
</tr>
<tr>
<td>Title</td>
<td>BPEL Worklist Analysis</td>
</tr>
<tr>
<td>Layout</td>
<td>Small</td>
</tr>
<tr>
<td>Show Worklist/Filter</td>
<td>Yes</td>
</tr>
<tr>
<td>Show Stations With Zero Count</td>
<td>Yes</td>
</tr>
<tr>
<td>Show Assigned Status</td>
<td>Yes</td>
</tr>
<tr>
<td>Show Completed Status</td>
<td>Yes</td>
</tr>
<tr>
<td>Show Expired Status</td>
<td>Yes</td>
</tr>
<tr>
<td>Show Inbound Status</td>
<td>Yes</td>
</tr>
<tr>
<td>Show Suspended Status</td>
<td>Yes</td>
</tr>
<tr>
<td>Show Un完成了 Status</td>
<td>Yes</td>
</tr>
<tr>
<td>Show Lists Requested Status</td>
<td>Yes</td>
</tr>
<tr>
<td>Locale Source</td>
<td>Portlet</td>
</tr>
<tr>
<td>Style Source</td>
<td>Portlet</td>
</tr>
</tbody>
</table>

*will be overridden by a valid setting of page/portlet parameter.

Table 21–3 shows the analysis portlet task filter and display customization options. The first part provides data filter customization options and the second part provides display customization options.

1. Enter the following parameter values.

Table 21–3  BPEL Worklist Analysis Portlet Filter and Display Customization Options
Step 4: Defining Portlet Parameters and Accessing Portlet Data Sources

Oracle BPEL Portlets

Table 21–3 (Cont.) BPEL Worklist Analysis Portlet Filter and Display Customization

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Category</td>
<td>Select one of the following:</td>
</tr>
<tr>
<td></td>
<td>■ My &amp; Group—tasks assigned to the user and the groups to which the logged-in user belongs</td>
</tr>
<tr>
<td></td>
<td>■ Reportees—tasks assigned to the users who report to the logged-in user</td>
</tr>
<tr>
<td></td>
<td>■ Owner—tasks that are owned by the logged-in user by way of process ownership</td>
</tr>
<tr>
<td></td>
<td>■ Creator—tasks that were created or initiated by the logged-in user</td>
</tr>
<tr>
<td></td>
<td>■ Previous—tasks that the logged-in user has updated</td>
</tr>
<tr>
<td>Created</td>
<td>Select the previous x days: (leave blank (that is, no restriction), 1 day, 7 days, 14 days, and 30 days)</td>
</tr>
<tr>
<td>Expiration</td>
<td>Select the next x days: (leave blank (that is, no restriction), 1 day, 7 days, 14 days, and 30 days)</td>
</tr>
<tr>
<td>Priority</td>
<td>Select from Any or 1 through 5, where 1 is the highest priority.</td>
</tr>
<tr>
<td>Business Process</td>
<td>Select one of the deployed business processes or Any.</td>
</tr>
<tr>
<td>Display Customization</td>
<td></td>
</tr>
<tr>
<td>Title</td>
<td>Select the portlet title.</td>
</tr>
<tr>
<td>Layout</td>
<td>Select the size of the portlet as a ratio of the maximum size (Small, Medium, Large, or Full).</td>
</tr>
<tr>
<td>Show Header/Footer</td>
<td>Set this flag to Yes (default option) if the header (category and user name) and footer (page generation time) are to be displayed.</td>
</tr>
<tr>
<td>Show Statuses with Zero Count</td>
<td>Select Yes to compact the portlet by skipping statuses for which there are no qualifying tasks</td>
</tr>
<tr>
<td>Group By</td>
<td>Select to group by Assignee, Business Process, Creator, or All (status summary of all tasks)</td>
</tr>
<tr>
<td>Show options for various statuses</td>
<td>Select only those statuses you want to see: Assigned, Completed, Expired, Errored, Suspended, Withdrawn, and Requested.</td>
</tr>
<tr>
<td>Locale Source</td>
<td>Select the locale for the portlet based on the specified source: Portlet (resolved from the browser and set language portlet) or Directory Service.</td>
</tr>
<tr>
<td>Style Source</td>
<td>Select the style (background color, font color, and font style) for the portlet based on the specified source: Portlet (based on page style used in the portal page) or Default (Oracle BPEL Control or Oracle BPEL Worklist Application style).</td>
</tr>
</tbody>
</table>

2. Click OK.

Analysis Portlet View

Figure 21–13 shows the analysis portlet for 10.1.2.0.2 and Figure 21–14 shows the analysis portlet for 10.1.3. The category of tasks and the current user name display at the top. The table displays the group by column, the corresponding statuses, and count. If there are multiple assignees (reportee category, for example), the status breakdown is repeated for each assignee. If the business process is chosen as the Group By column, the status breakdown is repeated for each deployed business process. The last page refreshed time displays at the bottom.
Step 5: Mapping Portlet Parameters with Page Parameters

Time-related properties of the portlets are also exposed as parameters and can be mapped to page parameters. This enables you to override the parameter values you set:

- In Step 1 on page 21-10 of “Defining Oracle BPEL Control Report Portlet Parameters and Accessing Portlet Data Sources”
- In Step 1 on page 21-15 of “Listing Portlet Customization” and Step 1 on page 21-18 of “Analysis Portlet Customization”

1. Click Page: Properties at the top of a report portlet. For this example, the Instance State report of the Oracle BPEL Control reports portlets is shown.
Step 5: Mapping Portlet Parameters with Page Parameters

Oracle BPEL Portlets

The Edit page appears.

2. Click the Parameters tab.
   a. If the Parameters tab does not display, return to the portlet reports page shown in Step 1 on page 21-20 to enable it.
   b. Click Page Group: Properties at the top of the report portlet.
   c. Click the Configure tab.
      The Edit Page Group page appears.
   d. Click Edit in the Parameters and Events section.

Parameters and Events
Click the Edit link to select whether to enable users to use parameters and events in the pages they create in this page group.

Page nhóm parameters and events are enabled. [Edit]

Approvals and Notifications

   e. Select Enable Parameters and Events.
   f. Click OK.
   g. Return to the Edit page shown in Step 1 on page 21-20.
   h. Click the Page: Properties tab.
   i. Click the Parameters tab on the Edit page.

Oracle BPEL Portlets 21-21
3. Add the page parameters and map the page parameters to the portlet parameters.
   - For Oracle BPEL Control reports portlets, the following page parameter properties are available:
     
     **Page Parameter Properties**
     Enter a display name, which identifies the parameter to other users. If you wish, enter a default value, or allow users to change the value of the parameter when they customize the page. Click the delete icon to remove the page parameter. The page parameter is displayed when users are customizing the page.
     
     | Name      | Display Name | Default Value |
     |-----------|--------------|---------------|
     | End_Date  | End_Date     | 7/29/05       |
     | Time_Interval | Time_Interval | HOURLY       |
     | End_Hour  | End_Hour     | 12            |
     | Number_of_Events | Number_of_Events | 10          |

     **Portlet Parameter Values**
     Expand a portlet to view its parameters and specify how to set the values of those parameters. You can choose parameters, system variables, or constant values.

     - For Oracle BPEL Worklist Application portlets, the following page parameter properties are available:
When you save these changes, the portlet picks up the property values from the page parameters instead of the ones set during the edit process.

Summary

This chapter describes how to deploy the Oracle BPEL Portlets and configure OracleAS Portal to provide access to data from these portlets.
This chapter provides an overview of Oracle BPEL Control reports. It provides descriptions of the types of reports you can create, how to create them, and how to analyze the results.

This chapter contains the following topics:

- Creating Oracle BPEL Control Reports
- Summary

Creating Oracle BPEL Control Reports

You can create reports in Oracle BPEL Control that enable you to:

- Receive an overall view of business process instance performance
- Analyze data for the BPEL process instances and make critical decisions
- Analyze data of the activities that constitute a business process
- Identify and debug faults and take appropriate corrective actions

Perform the following steps to create reports. The examples shown in this section describe how to create reports for the LoanFlowPlus sample BPEL process that is available under the `SOA_Oracle_Home\bpel\samples\demos\LoanDemoPlus\LoanFlowPlus` directory. To automatically build this sample in Oracle JDeveloper, create an application with an empty project and open the `LoanFlowPlus.jpr` file in the `LoanFlowPlus` directory.

1. Click a specific process to create reports for all its instances in the Deployed BPEL Processes column of the Dashboard tab of Oracle BPEL Control (for this example, LoanFlowPlus is selected).

2. Enter specific details to initiate an instance and click Post XML Message.

The Initiate tab appears by default.

2. Enter specific details to initiate an instance and click Post XML Message.
Creating Oracle BPEL Control Reports

3. Click Reports on the BPEL Processes tab.

The Reports tab appears:

This page enables you to generate several report types. The fields to complete in order to create reports are described in the following table:

<table>
<thead>
<tr>
<th>Page Element</th>
<th>Location in Page</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Report Type list</td>
<td>upper right</td>
<td>Select the type of report to create:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Process</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Performance</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Activity Sensor</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Fault Sensor</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ Process Time Distribution</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Descriptions of these report types are provided in the sections that follow.</td>
</tr>
<tr>
<td>End Date field</td>
<td>upper left</td>
<td>Enter the date at which to stop generating report data.</td>
</tr>
<tr>
<td>End Hour list</td>
<td>upper left</td>
<td>Select the hour at which to stop generating report data.</td>
</tr>
<tr>
<td>Time Interval list</td>
<td>upper middle</td>
<td>Select Daily, Hourly, or Weekly as the time interval for generating data.</td>
</tr>
<tr>
<td>Number of Time Intervals field</td>
<td>upper middle</td>
<td>Enter the time period for creating reports. This is the number of days, weeks, or hours (according to the Time Interval list selection) starting from the end date going backwards in time to determine the start date.</td>
</tr>
<tr>
<td>Go button</td>
<td>upper middle</td>
<td>Click to create the selected report type.</td>
</tr>
</tbody>
</table>

For example, if you enter 7/28/05 as the end date, 22:00 as the end hour, Weekly as the time interval, and 3 as the number of time intervals, the report is created for the three weeks between 7/08/05 at 10 PM and 7/29/05 at 10 PM.

4. See the following sections for details about creating the available report types:

■ Creating Process Reports
■ Creating Performance Reports
■ Creating Activity Sensor Reports
Creating Oracle BPEL Control Reports

- Creating Fault Sensor Reports
- Creating Process Time Distribution Reports

**Notes:**
- After creating any report type, you can save the report details to a comma-separated value (CSV) file by selecting Export to CSV. You can then review these reports in Microsoft Excel.
- You can also execute and view Oracle BPEL Control reports from Oracle Application Server Portal. See Chapter 21, "Oracle BPEL Portlets" for instructions.

**Creating Process Reports**

Process reports display data on the state of selected BPEL process instances and execution times of these instances. The data is grouped along the time line (X-Axis) into the number of time intervals based on the specified query criteria.

1. Select **Process** from the **Report Type** list.
2. Enter appropriate time period query details as described in the table in Step 3 on page 22-2 and click **Go**.
   
   Note that a progress bar and **Cancel** button display below the **Query** field during report creation. This occurs during the creation of all report types. If you want to stop report creation, click **Cancel**.

   The report results appear:

3. Review the **Instance State** section of the process report (the left side).

Oracle BPEL Control Reports 22-3
The graph shows the number of BPEL process instances initiated, faulted, successfully completed, and currently running. For this example, which uses the LoanFlowPlus sample BPEL process, the graph shows how many loans were requested (initiated), how many stopped abnormally due to faults in the system, how many were successfully processed, and how many are still pending.

4. Hold your cursor over a specific bar to display information. For this example, the information that displays identifies the series as faulted instances, the group (date) in which this information was gathered as 7/28/05, and the value (number) of faulted instances as four.

5. Click this bar to display the list of instances that faulted.
   This displays the list of four faulted instances on the Instances tab.

6. Click a specific instance from the list to debug.
   You can then debug the instance and take appropriate corrective actions.

7. Return to the process report results shown in Step 2 on page 22-3.

8. Review the Instance Execution Time section of the process report (the right side).
   This graph shows the maximum, minimum, and average execution times aggregated across all closed instances of the business process during the selected time interval. By default, process reports display all successfully and unsuccessfully completed processes (known as All Closed).

9. If you want to display only successfully completed (known as Closed Successfully) or unsuccessfully completed (known as Faulted) process instances, make an appropriate selection from the View Execution Time for list above the graph.
10. Click one of the minimum or maximum data points in the graph to show additional details. For example, click the maximum execution time to view information about the instance that took the longest amount of time to execute.

This displays the Instances tab for this instance.

11. Click the specific instance in the list. You can then debug the instance and take appropriate corrective actions.

Creating Performance Reports

Performance reports display the percentage of BPEL process instances that meet the Completion Time SLA field value. This enables you to identify bottlenecks in instance performance. You assign this value to the SLACompletionTime parameter in the Configurations tab of the Deployment Descriptor Properties window in Oracle JDeveloper. You specify an SLACompletionTime parameter value using the PnYT format. PnYT conforms to the PnMnDTnHnMnS convention, where:

- P (year)
- n (day)
- M (month)
- D (day)
- T (hour)
- H (hour)
- M (minute)
- S (second)
Creating Oracle BPEL Control Reports

- T is the date and time separator
- nY, nM, nD, nH, nM, and nS correspond to the number of years, months, days, hours, minutes, and seconds, respectively

For example, \( P0YT1.5S \) corresponds to 0 years, 1.5 seconds. The letter \( P \) and at least one unit of time must always be present.

If you define the completion time service level agreement (SLA) value for the LoanFlowPlus process as \( P0YT450S \) seconds (450 seconds), the report graph shows the percentage of instances that completed and did not complete within that time period.

This assignment adds this parameter to the \( \text{bpel.xml} \) file.

\[
<\text{configurations}>
<\text{property name}="\text{SLACompletionTime}"="P0YT450S"/>\n</\text{configurations}>
\]

1. Select Performance from the Report Type list.
2. Enter appropriate time period query details as described in the table in Step 3 on page 22-2 and click Go.

The report results appear:

This graph shows the percentage of instances that completed and did not complete within this time period.

3. Hold your cursor over a specific bar to display information. For this example, the information that displays below identifies the series of instances that satisfied the \( \text{SLACompletionTime} \) parameter value, the group (date) in which this information was gathered as 7/29/05, and information about the percentage and number of successfully completed instances and total number of instances for that day.
4. Click a specific bar to display the **Instances** tab. For example, click the red bar for instances that did not satisfy the SLA value. Holding your cursor over this bar before clicking displays specific information about these instances.

5. Click the specific instance in the list. You can then debug the instance and take appropriate corrective actions.

6. If you want to override the `SLACompletionTime` parameter value in the `bpel.xml` file, change the value in the **Completion Time SLA** field above the graph and click **Go** to re-execute the report.

The report results display details based on the new parameter value entered. Note that a **Default** link now displays to the right of the value you entered in Step 6.

7. Click **Default** to re-execute the report with the initial `SLACompletionTime` parameter value set during design time in the **Configurations** tab of the Deployment Descriptor Properties window.

**See Also:** Appendix C, “Deployment Descriptor Properties”

### Creating Activity Sensor Reports

Activity sensor reports show data for the activity sensors associated with the selected BPEL process over a specified time span. This enables you to analyze activity trends. Activity sensor reports show the activity name, sensor name, activity type, aggregated values of the minimum, maximum, and average execution time of the activity, and the number of occurrences of the activity. This report shows activity execution time only if activity sensors are defined for the BPEL process.

1. Select **Activity Sensor** from the **Report Type** list.

2. Enter appropriate time period query details as described in the table in Step 3 on page 22-2.

3. Make a selection from the **Activity Sensor** list. Only activities with sensors display in this list.

    - Select **All** to create a report for all activities in the BPEL process.
Creating Oracle BPEL Control Reports

- Select a specific activity for which to create a report.

4. Click Go.

   The report results appear (for this example, All was selected from the Activity Sensor list):

   ![Report Results](image)

   **Activity Sensors**
   - This report shows only activities for which activity sensors have been defined in the business process. Select an activity to view the variables that are defined within this activity sensor and the execution time graph of the activity.

   ![Activity Sensors](image)

   5. Click Show Details on the far left side to view additional information about an activity sensor.

   This displays details about all the variables associated with that activity sensor. It also shows a graph for the minimum, maximum, and average execution times, as collected by the chosen activity sensor over the selected time span.

   ![Activity Execution Time](image)

   6. Click the variable name in the Variable Name column (for this example, named input) to display the detailed values of the variable collected during the selected time period.
7. Click a specific instance ID on the Variable Details window to display the Flow diagram of that instance.

You can then debug the instance and take appropriate corrective actions.

Creating Fault Sensor Reports

Fault sensor reports show the data collected by fault sensors over a specified time interval. This enables you to analyze trends in faults while processing loan applications. The report shows the date and time of the fault occurrence, the activity in which the fault occurred, and a link to the faulted instance. For this example, the LoanFlowPlus sample BPEL process is used. There are two types of faults for which data is provided in this sample:

- Bankruptcy reports — the fault occurs when the customer has a negative credit rating
- The social security number of the customer requesting the loan was not found

1. Select Fault Sensor from the Report Type list.

2. Enter appropriate time period query details as described in the table in Step 3 on page 22-2 and click Go.

The report results display the types of fault messages.
Creating Oracle BPEL Control Reports

3. Click a specific instance ID in the **Instance ID** column to display the Flow diagram for the activity of the instance that faulted.

4. Debug the activity and take appropriate corrective actions.

**Creating Process Time Distribution Reports**

Process time distribution reports enable you to view the distribution of execution times across the activities of BPEL process instances. For example, you may have two activities, one for each loan application company returning a loan offer. You can see which activity is taking the longest amount of time to process the loan offer. You do not need to create sensors in activities to use this report. This report enables you to identify which activities are taking the most time to process.

1. Select **Process Time Distribution** from the **Report Type** list.
2. Enter appropriate time period query details as described in the table in Step 3 on page 22-2 and click **Go**.
The report results appear:

<table>
<thead>
<tr>
<th>Activity</th>
<th>Count</th>
<th>Average Execution Time (seconds)</th>
<th>Average Execution Time Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>assignProcess</td>
<td>10</td>
<td>7.44</td>
<td></td>
</tr>
<tr>
<td>assignProcessManual</td>
<td>10</td>
<td>8.68</td>
<td></td>
</tr>
<tr>
<td>copying</td>
<td>10</td>
<td>2.72</td>
<td></td>
</tr>
<tr>
<td>deploy2</td>
<td>10</td>
<td>8.68</td>
<td></td>
</tr>
<tr>
<td>writedatabase</td>
<td>10</td>
<td>7.50</td>
<td></td>
</tr>
<tr>
<td>initializeTask</td>
<td>10</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>initializeDB</td>
<td>10</td>
<td>5.00</td>
<td></td>
</tr>
<tr>
<td>initializeInstance</td>
<td>10</td>
<td>3.00</td>
<td></td>
</tr>
<tr>
<td>prepareWebApplication</td>
<td>10</td>
<td>1.00</td>
<td></td>
</tr>
<tr>
<td>prepareTask</td>
<td>10</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>prepareFile</td>
<td>10</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>readFile</td>
<td>10</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>receive2 (2011)</td>
<td>10</td>
<td>9.10</td>
<td></td>
</tr>
<tr>
<td>receive (187)</td>
<td>10</td>
<td>257.72</td>
<td></td>
</tr>
<tr>
<td>readInput</td>
<td>10</td>
<td>9.20</td>
<td></td>
</tr>
<tr>
<td>removeTextFile</td>
<td>10</td>
<td>104.50</td>
<td></td>
</tr>
</tbody>
</table>

3. Note which activities take the longest to complete.

4. Debug the activity and take appropriate corrective actions.

**Summary**

This chapter provides an overview of Oracle BPEL Control reports. It provides descriptions of the types of reports you can create, how to create them, and how to analyze the results.
This part provides reference details about troubleshooting issues, activities and services, deployment descriptor properties, and XPath extension functions.

This part contains the following appendices:

- Appendix A, "Troubleshooting and Workarounds"
- Appendix B, "BPEL Process Activities and Services"
- Appendix C, "Deployment Descriptor Properties"
- Appendix D, "XPath Extension Functions"
This appendix describes Oracle BPEL Process Manager troubleshooting methods.

This appendix contains the following topics:

- Troubleshooting General Issues
- Troubleshooting Sensors—the Custom Data Publisher
- Troubleshooting Oracle BPEL Worklist Application
- Summary

**Troubleshooting General Issues**

The following sections describe possible issues and solutions.

**Setting Properties for BPEL Processes to Successfully Complete and Catch Exception Errors**

The values to which you set the `transaction-timeout` and `syncMaxWaitTime` properties can impact whether a transaction scope successfully completes or times out and catches exception errors. For example, assume you have two processes:

- **TimeoutSubprocess** (A synchronous detail process that includes a wait activity set to three minutes)
- **TimeoutMainProcess** (An asynchronous main process that calls the TimeoutSubprocess)

If `syncMaxWaitTime` is set to 45 seconds (the default value) and `transaction-timeout` is set to 30 seconds, after 45 seconds the main process continues running and does not successfully complete and catch the following exception error as expected:

```
java.transaction.RollbackException: Timed out
```

In the `domain.log` file, the following exception error displays:

```
An exception occurred during transaction completion.; nested exception is:
java.transaction.RollbackException: Timed out
```

Perform the following procedures for the main process to successfully complete and catch the exception error:

1. Set the `transaction-timeout` and `syncMaxWaitTime` properties as follows:
Troubleshooting General Issues

This causes the main process to successfully complete and catch the exception error.

Developer Prompt on Windows 2000

The developer prompt on Windows 2000 can fail to display when selecting Start > All Programs > Oracle_Home > Oracle BPEL Process Manager > Developer Prompt. This is a known classpath length issue with Windows 2000. For the developer prompt to successfully display, you must shorten your classpath. For example, when you install Oracle BPEL Process Manager, limit the length of the directory path of your Oracle home.

See Also: “Starting Oracle BPEL Process Manager Components” on page 2-2

Correcting Validation Errors in Complex Processes

If you have complex processes with validation errors (for example, assign activities with multiple copy rules that are embedded inside several scopes), the recommended method for accessing and correcting these errors is as follows:

1. Right-click the error and select Go to Source to access the source code that errored.
2. Review the source code to identify the error.
3. Click Diagram and go to the graphical view of the error.
4. Make corrections in the graphical view.

Handling Long-Running Processes

When a process has a long-standing activity and the server timeout value is set to less than the time that has elapsed since the previous dehydration point has been reached,
you can see exception messages similar to the following in the Oracle BPEL Server window:

```
Message handle error.
An exception occurred while attempting to process the message
  "cm:collaxa:cube:engine:dispatch:message:invoke:invokeInstance
  (server)=message": the exception is: Transaction was rolled back: timed out;
  nested exception is: java.rmi.RemoteException:
  No Exception - originate from:java.lang.Exception: No Exception - originate
  from:; nested exception is:
  java.lang.Exception: No Exception - originate from:
```

As a workaround, increase the `transaction-config timeout` value in the `transaction-manager.xml` file. For example:

```xml
<transaction-config timeout="30000"/>
```

The location of this file depends on the method by which you installed Oracle BPEL Process Manager:

- For Oracle Application Server SOA Basic installations, the file is located in `SOA_Oracle_Home\j2ee\home\config`.
- For Oracle BPEL Process Manager installations, the file is located in `SOA_Oracle_Home\bpel\system\appserver\oc4j\j2ee\home\config`.

See Oracle BPEL Process Manager Installation Guide for a detailed description on setting this parameter.

### Creating an Empty BPEL Process and Importing a Schema

If you create an empty project (which is common with adapter endpoint projects) and import an XSD file, you cannot deploy the project without editing the `project_name.bpfl` file.

As a workaround, perform the following steps:

1. Create an asynchronous project.
2. Leave the client partner link alone (it enables you to import XSDs because it refers to the `process_name.wsdl` file that has the necessary imports).
3. Edit the `receiveInput` receive activity of the client partner link and have it point to the new adapter inbound endpoint in the `Partner Link` field.

### Troubleshooting Sensors—The Custom Data Publisher

The following sections describe possible issues and solutions.

#### Data Publisher Is Not Working

**Problem**
The custom data publisher is not working.

**Solution**
- Make sure that the `class` file has been generated and that it is in the system classpath. See the `obsetenv.bat` file for this definition, or the BPEL suitcase.
- Ensure that you have implemented the data publisher interface.
Troubleshooting Sensors—The Custom Data Publisher

- If you compile your data publisher into the system classpath, then you must
  restart Oracle BPEL Process Manager. You may have made changes to the data
  publisher without restarting Oracle BPEL Process Manager.
- It is possible that an exception is being thrown in your data publisher. Check the
  log file for any exceptions, or temporarily add a try/catch block around all your
  code. In the catch, print the stack trace. These messages display on the text
  window that opens when you start Oracle BPEL Process Manager.

Data Publisher Works, But Business Process Runs Slowly

  Problem
  The data publisher works fine, but the business process runs very slowly.
  
  Solution
  There are a couple of options.
  First, you can attempt to profile your code. The do-user-sensor-callback statistic in
  Oracle BPEL Control records how much time is spent publishing sensor data.
  Second, you can switch from a custom data publisher to a JMS Publisher. Then, you
  can deploy a message-driven bean to the application server to publish data whenever
  data is sent to that particular JMS destination. This decouples data publishing from
  process execution.

Caching Data in the Data Publisher Is Not Supported

  Problem
  To improve performance, I want to cache data in my data publisher. Is this supported?
  
  Solution
  This is not supported. Data publishers must be stateless.

Unexpected Errors in the Data Publisher

  Problem
  My data publisher works fine most of the time, but sometimes I get a weird error.
  It is possible that your data publisher is experiencing concurrency issues.
  
  Solution
  Data publishers must be coded in a thread-safe manner. This means that the Java code
  must be thread safe as well as the utilization of resources, such as databases or files.

Data Extracted to XML Is Difficult to Work With

  Problem
  The data I extract is complex XML. It is difficult to work with. Can I do anything to
  make it simpler?
Solution
While the W3C DOM model is somewhat cumbersome, there are third-party models (such as DOM4J) that make things easier. It is easy to create a DOM4J object from its corresponding W3C DOM object. Another option is to generate JAXB objects or schema objects for the data you extract. Then you can use the generated Java classes to manipulate data more easily.

Troubleshooting Oracle BPEL Worklist Application
The following sections describe possible issues and solutions.

Not Able to Log In to the Worklist Application
You cannot log in to the Worklist Application if your information is not available in the identity service. Check with an administrator to verify that your user information is present in the identity system (a file based on LDAP, such as Oracle Internet Directory).

Information Is Displayed in a Different Language
The Worklist Application gets a user’s language (locale) preferences from the identity service and displays the information in that locale. If information is displayed in the wrong language, make sure that the user’s preference is set to a supported locale. See "Accessing the Worklist Application in Local Languages" on page 16-39 for more information.

Dates and Times Are Displayed Incorrectly
The Worklist Application gets a user's timezone preference from the identity service and displays the information in this timezone. Also, the date and time is formatted to suit the language (locale) preference. Make sure that these preferences are correctly specified in the identity service.

The User Is Not Permitted to Perform an Action
You may see an error message that says something like:

*User jcooper is not permitted to perform the action Update on task Loan application for John with id....*

Check if the user has permission to perform the action or if the action can be performed on the task in its current state. You can also check for the following:

- The task expired between the time the user loaded the page and actually performed the action.
- The task was updated by another user (such as a manager, owner, or administrator) between the time the user loaded the page and actually performed the action.

Expected Task Is Not Listed Under Task Titles
On the Worklist Application home page, under the Title column, if you do not see a task listed that you expected to see, then it may have been modified by another user or by the system.
Another user, such as a manager or group member, may have modified the task by performing any of the following actions:
- Complete
- Suspend
- Request More Information
Also, the filer of the task may have withdrawn (cancelled) the task.
The system can modify a task in the following situations:
- If the process instance associated with a task was purged or archived, the task is also purged or archived and may not be accessible.
- If a task expires
- If a task encounters a system error such as an incorrect assignee
In most of the preceding cases, you can view the task by changing the filters to a broader category (such as Any or All).

**Summary**

This appendix describes Oracle BPEL Process Manager troubleshooting methods.
This appendix describes the activities and services that you use when designing a BPEL process in Oracle JDeveloper.

This appendix contains the following topics:
- Process Activities Overview
- Services Overview
- Validation When Loading a Process Diagram
- Summary

**Process Activities Overview**

Oracle JDeveloper includes activities that are available for dragging and dropping into a BPEL process. These activities enable you to perform specific tasks within a process. This section provides a brief overview of these activities and provides references to other documentation that describes how to use these activities.

This section contains the following topics:
- Tabs Common to Many Activities
- Assign Activity
- Compensate Activity
- Decide Activity
- Email Activity
- Empty Activity
- Fax Activity
- Flow Activity
- FlowN Activity
- Human Task Activity
- Invoke Activity
- Java Embedding Activity
- Pager Activity
- Pick Activity
- Receive Activity
Process Activities Overview

- Reply Activity
- Scope Activity
- Sequence Activity
- SMS Activity
- Switch Activity
- Terminate Activity
- Throw Activity
- Transform Activity
- Voice Activity
- Wait Activity
- While Activity

See Also: The following documentation for additional details about activities:
- Oracle BPEL Process Manager Quick Start Guide
- Oracle BPEL Process Manager Order Booking Tutorial
- Business Process Execution Language for Web Services Specification
- SOA_Oracle_Home\bpel\samples\references directory

Tabs Common to Many Activities

While each activity performs specific tasks, many activities include tabs that enable you to perform similar tasks. This section describes these common tabs.

- The Sensors tab displays on all activities and enables you to create sensors for capturing details about an activity.
- The Correlations tab displays in invoke, receive, and reply activities, the onMessage branch of pick activities, and the OnMessage variant of event handlers. Correlation sets address complex interactions between a process and its partners by providing a method for explicitly specifying correlated groups of operations within a service instance. A set of correlation tokens is defined as a set of properties shared by all messages in the correlated group.
- The Adapters tab displays in invoke, receive, and reply activities, and the onMessage branch of pick activities. You create header variables for use with the Advanced Queuing (AQ), File, FTP, MQ, and Java Message Service (JMS) adapters.
- The Annotations tab displays on all activities and enables you to provide descriptions in activities in the form of code comments and name and pair value assignments.

Note the following issues when using annotations in Oracle JDeveloper:
- The Annotations tab in activities of Oracle JDeveloper does not provide a method for changing the order of annotations. As a workaround, change the order of annotations in the Source view of the project’s BPEL file in Oracle JDeveloper.
The otherwise branch in a switch activity does not allow you to create annotations. However, the case branch in a switch activity does provide this functionality.

**See Also:**
- The Online help for these tabs for additional details about their use
- "Using Correlation Sets in an Asynchronous Service" on page 5-14
- Chapter 17, "Sensors"
- Oracle Adapters for Files, FTP, Databases, and Enterprise Messaging User's Guide

### Assign Activity

This activity provides a method for data manipulation, such as copying the contents of one variable to another. This activity can contain any number of elementary assignments.

When you double-click the Assign icon, the Assign window appears. You can perform the following tasks:

- Click the General tab to provide the assign activity with a meaningful name.
- Click the Copy Operation tab and the Create icon (shown in Figure B–1), and then select Copy Operation, to access the Create Copy Operation window. Other selections such as Append Operation and Insert-After Operation are also available. This enables you to copy the contents of the source element (variable, expression, XML fragment, or partner link) in the From field to the contents of the destination element in the To field. You can also select a part (typically the payload) and an XPath query (a language for addressing parts of an XML document).
If an assign activity contains more than one `bpelx:append` setting, it must be split into two assign activities. Otherwise, Oracle JDeveloper moves the `bpelx:append` to the end of the list each time, which can cause problems. As a workaround, move it manually.

See Also: The following documentation for examples of using the assign activity:
- Chapter 3, "Manipulating XML Data in BPEL"
- Oracle BPEL Process Manager Order Booking Tutorial
- Oracle BPEL Process Manager Quick Start Guide
- SOA_Oracle_Home\bpel\samples\references\Assign

Compensate Activity

This activity invokes compensation on an inner scope activity that has already successfully completed. This activity can be invoked only from within a fault handler or another compensation handler. Compensation occurs when a process cannot complete several operations after already completing others. The process must return and undo the previously completed operations. For example, assume a process is designed to book a rental car, a hotel, and a flight. The process books the car and the hotel, but is unable to book a flight for the correct day. In this case, the process performs compensation by unbooking the car and the hotel.

The compensation handler is invoked with the compensate activity, which names the scope on which the compensation handler is to be invoked.
When you double-click the **Compensate** icon, the Compensate window shown in Figure B–2 appears. You can perform the following tasks:

- Click the **General** tab to provide the activity with a meaningful name and select the scope activity on which the compensation handler is to be invoked.

![Figure B–2 Compensate Activity](image)

**See Also:**
- "BankTransferDemo" on page 1-7 for a demonstration that uses a compensate activity
- "Using Compensation After Undoin a Series of Operations" on page 8-9

**Decide Activity**

This activity enables you to define a process that invokes the decision service partner link that you created with the Decision Service Wizard. This activity also enables you to create copy operation assignments between the fact data in your rule set or function and BPEL variables.

When you drag and drop the **Decide** icon, the Decide window appears. You provide the following information, as shown in Figure B–3.
Enter a name, select the decision service partner link you created, and the operation to perform (invocation pattern). If you have not created a decision service, click the first icon to the right of the Decision Service field.

Click Assign Input Facts, then click Create to create mappings for the input facts. This enables you to assign BPEL variables to the facts to be asserted or to the function input parameters. This enables you to create assignments that map BPEL input variables to automatically created BPEL variables that correspond to the input (assert) fact type.

If you selected an invocation pattern that retrieves results, click Assign Output Facts, then click Create to create mappings for the output facts. This enables you to assign values from a function return value or rule set result to a BPEL variable. This enables you to create assignments that map automatically created BPEL variables that correspond to the output (watch) fact type.

A decide activity consisting of assign and invoke activities to the decision service partner link is created after you click OK.

See Also: "Decide Activity" on page 18-14

Email Activity
This activity enables you to send an e-mail notification about an event. For example, an online shopping business process of an online bookstore sends a courtesy e-mail message to you after the items are shipped. The business process calls
the notification service with your user ID and notification message. The notification service gets the e-mail address from Oracle Internet Directory.

When you drag and drop an Email icon into a BPEL process, the Edit Email window shown in Figure B–4 appears.

**Figure B–4  Edit Email Window**

Use the Edit Email dialog to enter notification details, as described in Table B–1.

**Table B–1  E-mail Notification Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>From Account</td>
<td>The name of the account used to send this message. The configuration details for this e-mail account name must exist on Oracle BPEL Server.</td>
</tr>
<tr>
<td>To</td>
<td>The e-mail address to which the message is to be delivered. This can be a) a static e-mail address entered at the time the message is created, or b) an e-mail address looked up using the identity service, or c) a dynamic address from the payload. The XPath Expression Builder can be used to get the dynamic e-mail address from the input. See ’Setting E-mail Addresses and Telephone Numbers Dynamically’ on page 14-13.</td>
</tr>
</tbody>
</table>
Empty Activity

This activity enables you to insert a no-operation instruction into a process. This activity is useful when you need to use an activity that does nothing (for example, when a fault needs to be caught and suppressed). Figure B–5 shows the empty activity.

Table B–1 (Cont.) E-mail Notification Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CC and Bcc</td>
<td>The e-mail addresses to which the message is copied and blind copied. This can be a static or dynamic address as described for the To address.</td>
</tr>
<tr>
<td>Reply To</td>
<td>The e-mail address to use for replies. This can be a static or dynamic address as described for the To address.</td>
</tr>
<tr>
<td>Subject</td>
<td>Subject of the e-mail message. This can be free text or dynamic text. The XPath Expression Builder can be used to set dynamic text based on data from process variables that you specify.</td>
</tr>
<tr>
<td>Body</td>
<td>Message body of the e-mail message. This can be plain text, XML, free text, or dynamic text, as described for the Subject parameter.</td>
</tr>
<tr>
<td>Multipart message with attachments</td>
<td>Select to specify e-mail attachments. See “Setting E-mail Attachments” on page 14-5. The number of attachments if Multipart message is selected. The number includes the body. For example, if you have a body and one attachment, specify 2 here.</td>
</tr>
</tbody>
</table>

See Also:

- Oracle BPEL Process Manager Order Booking Tutorial for an example of using an email activity
- “The E-mail Notification Channel” on page 14-4
**Fax Activity**

This activity enables you to send a fax notification about an event.

When you drag and drop a Fax icon into a BPEL process, the Edit Fax window shown in Figure B–6 appears.

See Also:  "Using the Empty Activity to Insert No-Op Instructions into a Business Process" on page 8-8
Use the Edit Fax dialog to enter notification details, as described in Table B–2.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fax Number</td>
<td>The fax number to which the message is to be delivered. This can be a) a static fax number entered at the time the message is created, or b) a fax number looked up using the identity service, or c) a dynamic fax number from the payload. The XPath Expression Builder can be used to get the dynamic fax number from the input.</td>
</tr>
<tr>
<td>Cover Page</td>
<td>The cover page name. The cover page details must exist on the server. The cover page can be in PDF, Microsoft Word, HTML, or plain text format. (This field is optional.) The XPath Expression Builder can be used to set dynamic text based on data from process variables that you specify.</td>
</tr>
<tr>
<td>Body</td>
<td>Fax message body. This must be plain text or HTML. This can be free text or dynamic text as described for the Cover page parameter.</td>
</tr>
</tbody>
</table>

See Also: "The Fax Notification Channel" on page 14-8
Flow Activity

This activity enables you to specify one or more activities to be performed concurrently. A flow activity completes when all activities in the flow have finished processing. Completion of a flow activity includes the possibility that it can be skipped if its enabling condition is false.

For example, assume you use a flow activity to enable two loan offer providers (United Loan service and Star Loan service) to start in parallel. In this case, the flow activity contains two parallel activities – the sequence to invoke the United Loan service and the sequence to invoke the Star Loan service. Each service can take an arbitrary amount of time to complete their loan processes.

Figure B–7 shows an initial flow activity with its two panels for parallel processing. You drag and drop activities into both panels to create parallel processing. When complete, a flow activity looks like that shown in Figure B–8.

*Figure B–7  Flow Activity (At Time of Creation)*

*Figure B–8  Flow Activity (After Design Completion)*
Process Activities Overview

**FlowN Activity**

This activity enables you to create activities within a flow. You specify the number of branches of these activities to create.

*Figure B–9 shows a FlowN activity.*

**Human Task Activity**

This activity enables you to describe the tasks, input or output information, and procedural steps performed by users or groups as part of the end-to-end business process. For example, an insurance company can design a workflow application to ensure that a claim is handled consistently from initial call to final settlement. The workflow application ensures that each person handling the claim uses the correct

---

**See Also:** The following documentation for examples of using the flow activity:

- "Defining a Parallel Flow" on page 6-2
- Oracle BPEL Process Manager Order Booking Tutorial
- SOA_Oracle_Home\bpel\samples\references\Flow

**See Also:**

- "Customizing the Number of Flow Activities by Using the flowN Activity" on page 6-4
- SOA_Oracle_Home\bpel\samples\references\FlowN
online form and successfully completes their step before enabling the process to proceed to the next person and procedural step.

You create a workflow to manage and enforce the consistent handling of work. After creation, you can also modify your workflow. At run time, the workflow results in the creation of tasks that can be accessed through the Oracle BPEL Worklist Application.

When you drag and drop a Human Task icon, the Add a Human Task window shown in Figure B–10 appears.

**Figure B–10  Human Task Activity**

Create a task definition by clicking the second icon to the right of the Task Definition field. The Human Task editor opens. This is where you create the human workflow task, as shown in Figure B–11. Your inputs are saved in the `human_task_name.task` file.
Invoke Activity

This activity enables you to specify an operation you want to invoke for the service (identified by its partner link). The operation can be one-way or request-response on a port provided by the service. You can also automatically create variables in an invoke...
activity. An invoke activity invokes a synchronous service or initiates an asynchronous Web service.

The invoke activity opens a port in the process to send and receive data. It uses this port to submit required data and receive a response. For synchronous callbacks, only one port is needed for both the send and the receive functions.

When you double-click the **Invoke** icon, the Invoke window shown in Figure B–12 appears. You can perform the following tasks:

- Provide the activity with a meaningful name.
- Select the partner link for which to specify an operation
- Select the operation to be performed
- Automatically create a variable or select an existing variable in which to transport the data (payload)

*Figure B–12  Invoke Activity*
Java Embedding Activity

This activity enables you to add custom Java code to a BPEL process using the Java BPEL exec extension `<bpelx:exec>`. This is useful when you already have Java code that can perform a function, and want to use this existing code instead of starting over.

When you double-click this activity, the Edit Java Embedding window shown in Figure B–13 appears.

Figure B–13 Java Embedding Activity

If you use this activity, ensure that you add the JAR files to the Oracle JDeveloper classpath or put them in the `JDev_Oracle_Home\jdev\lib\ext` directory to ensure that your project compiles properly during design time.

In addition, place the corresponding class files in the `SOA_Oracle_Home\bpel\system\classes` directory.

Perform the following Java JAR file configuration steps to ensure correct run time behavior. This example describes the configuration steps for Windows operating systems. Perform similar steps for UNIX operating systems.

1. Add the files into the BPEL client `bpelc` CLASSPATH:
   - Open `SOA_Oracle_Home\bpel\bin\obsetenv.bat`. For UNIX operating systems, the file is named `obsetenv.sh`.
   - Edit the file as follows:
@REM set MY_CLASSES_DIR=%CLASSPATH%
set MY_CLASSES_DIR=%OB_HOME%/system/classes
set MY_CLASSPATH=%MY_CLASSES_DIR%;your_jar_files

2. Add the files into the Oracle BPEL Server bpelc CLASSPATH in either of two ways:
   a. Open SOA_Oracle_Home/bpel/domains/default/config/domain.xml (if the default domain is being used).
   b. Edit the file as follows:
      <property id="bpelcClasspath">
      <name>BPEL process compiler classpath</name>
      <value>your_jar_files;...</value>
      <comment>...</comment>
      </property>
   c. Restart Oracle BPEL Server.
   or
   a. Go to Manage BPEL Domain > Configuration on Oracle BPEL Control.
   b. Go to the bpelcClasspath property.
   c. Enter your_jar_files; in the Value field.
   d. Click Apply.
   e. Restart Oracle BPEL Server.

3. Add the files into the Oracle Application Server CLASSPATH:
   a. Open application.xml, which is in SOA_Oracle_Home/j2ee/home/config
   b. Edit the file as follows:
      <library path="your_jar_files"/>
      <library path="C:\Ora_Home\bpel\system\classes"/>

See Also: “Using Java Embedding in a BPEL Process” on page 9-8

Pager Activity

This activity enables you to send a pager notification about an event.
When you drag and drop a Pager icon into a BPEL process, the Edit Pager window shown in Figure B–14 appears.
Use the Edit Pager dialog to enter notification details, as described in Table B–3.

Table B–3  Pager Notification Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>From Number</td>
<td>The pager number from which the message is to be sent. This can be: a) a static pager number entered at the time the message is created, or b) a dynamic pager number from the payload. The XPath Expression Builder can be used to get the dynamic pager number from the input.</td>
</tr>
<tr>
<td>Pager Number</td>
<td>The number of the recipient of this message. This can be: a) a static pager number entered at the time the message is created, or b) a pager number looked up using the identity service, or c) a dynamic pager number from the payload. The XPath Expression Builder can be used to get the dynamic pager number from the input.</td>
</tr>
<tr>
<td>Body</td>
<td>Pager message body. This must be plain text. This can be free text or dynamic text as described for the From Number parameter.</td>
</tr>
</tbody>
</table>

See Also:  “The Pager Notification Channel” on page 14-10
Pick Activity

This activity waits for the occurrence of one event in a set of events and performs the activity associated with that event. The occurrence of the events is often mutually exclusive (the process either receives an acceptance or rejection message, but not both). If more than one of the events occurs, then the selection of the activity to perform depends on which event occurred first. If the events occur nearly simultaneously, there is a race and the choice of activity to be performed is dependent on both timing and implementation.

The pick activity provides two branches, each one with a condition. When you double-click the Pick icon, the pick activity shown in Figure B–15 appears and displays these two branches: onMessage (on the left) and onAlarm (on the right). The onMessage branch contains the code for receiving a reply, for example, from a loan service. The onAlarm branch contains the code for a timeout, for example, after one minute. Whichever branch completes first is executed; the other branch is not. The branch that has its condition satisfied first is executed.

If you add correlations to an OnMessage branch of a Pick activity in Oracle JDeveloper, the correlations syntax is placed after the assign activity syntax. The correlation syntax must go before the assign activity.

As a workaround, perform the following steps:

1. Create a correlation set in Oracle JDeveloper.
2. Assign this to the OnMessage branch.
3. Complete the remaining design tasks.
4. Before making or deploying the BPEL process, move the correlation syntax before the assign activity in the BPEL source code.
Receive Activity

This activity specifies the partner link from which to receive information and the port type and operation for the partner link to invoke. This activity waits for an asynchronous callback response message from a service, such as a loan application approver service. While the BPEL process is waiting, it is dehydrated (compressed and stored) until the callback message arrives. The contents of this response are stored in a response variable in the process.

When you double-click the Receive icon, the Receive window shown in Figure B–16 appears. You can perform the following tasks:

- Provide the receive activity with a meaningful name.
- Select the partner link service for which to specify an operation
- Select the operation to be performed
- Automatically create a variable or select an existing variable in which to transport the callback response

See Also:

- "107. Exceptions" on page 1-11 for a tutorial that uses a pick activity
- "108. Timeouts" on page 1-11 for a tutorial that uses a pick activity
- "Using the Pick Activity to Select Between Continuing a Process or Waiting" on page 10-2
- "Defining a Timeout" on page 10-4
- Chapter 12, "Interaction Patterns"
- SOA_Oracle_Home\bpel\samples\references\Pick for an example of using the pick activity
This activity allows the process to send a message in reply to a message that was received through a receive activity. The combination of a receive activity and a reply activity forms a request-response operation on the WSDL port type for the process.

Figure B–17 shows the reply activity.
This activity consists of a collection of nested activities that can have their own local variables, fault handlers, compensation handlers, and so on. A scope activity is analogous to a `{ }` block in a programming language.

Each scope has a primary activity that defines its behavior. The primary activity can be a complex structured activity, with many nested activities within it to arbitrary depth. The scope is shared by all the nested activities.

When you double-click the Scope icon, the Scope window shown in Figure B–18 appears. Define appropriate activities inside the scope activity.
Fault handling is associated with a scope activity. The goal is to undo the incomplete and unsuccessful work of a scope activity in which a fault has occurred. You define catch activities in a scope activity to create a set of custom fault-handling activities. Each catch activity is defined to intercept a specific type of fault.

Figure B-19 shows the Add Catch Branch icon inside a scope activity. Figure B-20 shows the catch activity area that appears when you click the Add Catch Branch icon. Within the area defined as Drop Activity Here, you drag and drop additional activities to create fault handling logic to catch and manage exceptions.

For example, a client provides a social security number to a Credit Rating service when applying for a loan. This number is used to perform a credit check. If a bad credit history is identified or the social security number is identified as invalid, an assign activity inside the catch activity notifies the client of the loan offer rejection. The entire loan application process is terminated with a terminate activity.
Sequence Activity

This activity enables you to define a collection of activities to be performed in sequential order. For example, you may want the following activities performed in a specific order:

- A customer request is received in a receive activity.

See Also: The following documentation for examples of using the scope activity and fault handling:

- "Using the Scope Activity to Manage a Group of Activities" on page 8-6
- "Using a Fault Handler within a Scope" on page 8-8
- Chapter 12, "Interaction Patterns"
- Oracle BPEL Process Manager Order Booking Tutorial
- SOA_Oracle_Home\bpel\samples\references\Catch
The request is processed inside a flow activity that enables concurrent behavior.

A reply message with the final approval status of the request is sent back to the customer in a reply activity.

A sequence activity makes the assumption that the request can be processed in a reasonable amount of time, justifying the requirement that the invoker wait for a synchronous response (because this service is offered as a request-response operation). When this assumption cannot be made, it is better to define the customer interaction as a pair of asynchronous message exchanges.

When you double-click the Sequence icon, the activity area shown in Figure B–21 appears. Define appropriate activities inside the sequence activity.

Figure B–21 Sequence Activity

See Also:

- "Defining a Parallel Flow" on page 6-2
- Chapter 12, "Interaction Patterns"

SMS Activity

This activity enables you to send a short message system (SMS) notification about an event.

When you drag and drop an SMS icon into a BPEL process, the Edit SMS window shown in Figure B–22 appears.
Use the Edit SMS dialog to enter notification details, as described in Table B–4.

**Table B–4 SMS Notification Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>From number</td>
<td>The telephone number from which to send the SMS notification. This can be a static telephone number entered at the time the message is created or a dynamic telephone number from the payload. See &quot;Setting E-mail Addresses and Telephone Numbers Dynamically&quot; on page 14-13.</td>
</tr>
<tr>
<td>Telephone number</td>
<td>The telephone number to which the message is to be delivered. This can be a) a static telephone number entered at the time the message is created, or b) a telephone number looked up using the identity service, or c) a dynamic telephone number from the payload. The XPath Expression Builder can be used to get the dynamic telephone number from the input.</td>
</tr>
<tr>
<td>Subject</td>
<td>Subject of the SMS message. This can be free text or dynamic text. The XPath Expression Builder can be used to set dynamic text based on data from process variables that you specify.</td>
</tr>
<tr>
<td>Body</td>
<td>SMS message body. This must be plain text. This can be free text or dynamic text as described for the Subject parameter.</td>
</tr>
</tbody>
</table>
Switch Activity

This activity consists of an ordered list of one or more conditional branches defined in a case branch, followed optionally by an otherwise branch. The branches are considered in the order in which they appear. The first branch whose condition is true is taken and provides the activity performed for the switch. If no branch with a condition is taken, then the otherwise branch is taken. If the otherwise branch is not explicitly specified, then an otherwise branch with an empty activity is assumed to be available. The switch activity is complete when the activity of the selected branch completes.

A switch activity differs in functionality from a flow activity. For example, a flow activity enables a process to gather two loan offers at the same time, but does not compare their values. To compare and make decisions on the values of the two offers, a switch activity is used. The first branch is executed if a defined condition (inside the case branch) is met. If it is not met, the otherwise branch is executed.

Figure B–23 shows a switch activity with the following defined branches.

See Also: “The SMS Notification Channel” on page 14-11

Terminate Activity

A terminate activity enables you to end the tasks of an activity (for example, the fault handling tasks in a catch branch). For example, if a client’s bad credit history is identified or a social security number is identified as invalid, a loan application process is terminated, and the client’s loan application document is never submitted to the service loan providers.

Figure B–24 shows a terminate activity in the otherwise branch of a switch activity.

See Also: The following documentation for examples of using the switch activity:
- “Using a Switch Activity to Define Conditional Branching” on page 7-2
- Chapter 12, “Interaction Patterns”
- Oracle BPEL Process Manager Order Booking Tutorial
- SOA_Oracle_Home\bpel\samples\references\Switch
Figure B–24  Terminate Activity

This activity generates a fault from inside the business process. When you double-click the **Throw** icon, the Throw window shown in Figure B–25 appears.

See Also: The following documentation for examples of using the terminate activity:
- “Using the Terminate Activity to Stop a Business Process Instance” on page 8-10
- Oracle BPEL Process Manager Order Booking Tutorial
- SOA_Oracle_Home\bpe\samples\references\Terminate

Throw Activity

This activity generates a fault from inside the business process. When you double-click the **Throw** icon, the Throw window shown in Figure B–25 appears.
Figure B–25  Throw Activity

This activity enables you to create a transformation that maps source elements to target elements (for example, incoming purchase order data into outgoing purchase order acknowledgment data).

When you double-click the Transform icon, the Transform window shown in Figure B–26 appears. This window enables you to perform the following tasks:

- Define the source and target variables and parts to map
- Specify the transformation mapper file
- Click the second icon (the Create Mapping icon) to the right of the Mapper File field to access a window for graphically mapping source and target elements. This window enables you to drag and drop (map) a source element to a target element.

See Also: The following documentation for examples of using the throw activity:
- "Throwing Internal Faults" on page 8-6
- SOA_Oracle_Home\bpel\samples\references\Throw

Transform Activity
Voice Activity

This activity enables you to send a telephone voice notification about an event.

When you drag and drop a Voice icon into a BPEL process, the Edit Voice window shown in Figure B–27 appears.

See Also: The following documentation for examples of using the transform activity:

- Chapter 13, "XSLT Mapper and Transformations"
- Oracle BPEL Process Manager Order Booking Tutorial
- SOA_SOA_Home\bpe1\examples\demos\XSLMapper
Use the Edit Voice dialog to enter notification details, as described in Table B–5.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Telephone number</td>
<td>The telephone number to which the message is to be delivered. This can be a) a static telephone number entered at the time the message is created, or b) a telephone number looked up using the identity service, or c) a dynamic telephone number from the payload. The XPath Expression Builder can be used to get the dynamic telephone number from the input.</td>
</tr>
<tr>
<td>Body</td>
<td>Message body. This can be plain text or XML. Also, this can be free text or dynamic text. The XPath Expression Builder can be used to set dynamic text based on data from process variables that you specify.</td>
</tr>
</tbody>
</table>

See Also:  “Voice Activity” on page B-30

**Wait Activity**

This activity allows a process to specify a delay for a certain period of time or until a certain deadline is reached. A typical use of this activity is to invoke an operation at a
certain time. This activity allows you to wait for a given time period or until a certain time has passed. Exactly one of the expiration criteria must be specified. When you double-click the Wait icon, the Wait window shown in Figure B–28 appears.

**Figure B–28  Wait Activity**

![Wait Activity Window](image)

See Also: The following documentation for examples of using the wait activity:

- “Using the Wait Activity to Set an Expiration Time” on page 10-4
- Oracle BPEL Process Manager Order Booking Tutorial
- SOA_Oracle_Home\bpel\samples\references\Wait

**While Activity**

This activity supports repeated performance of a specified iterative activity. The iterative activity is repeated until the given while condition is no longer true. When you double-click the While icon, the While window shown in Figure B–29 appears. You can enter expressions in this window.
Services Overview

With the following services, BPEL processes can communicate with Web-based applications and clients.

See Also:
- Oracle Adapters for Files, FTP, Databases, and Enterprise Messaging User's Guide for more information about the adapters described in the following sections
- Oracle Application Server Adapter Concepts for conceptual information
- Oracle BPEL Process Manager Order Booking Tutorial for examples using adapters

This section contains the following topics:
- AQ Adapter
- Database Adapter
## Services Overview

- Decision Service
- EJB Web Service
- File Adapter
- FTP Adapter
- Java Web Service
- JMS Adapter
- MQ Adapter
- Oracle Applications
- PartnerLink

### AQ Adapter

This adapter acts as both a dequeue (inbound) and enqueue (outbound) messaging adapter. In the inbound direction, the adapter polls the queues for messages to dequeue from a destination. In the outbound direction, the adapter enqueues messages to the queue for subscribers to dequeue.

**See Also:** The following documentation for examples of using this adapter:

- SOA_Oracle_Home\bpel\samples\tutorials\124.AQAdapter

### Database Adapter

This adapter enables a BPEL process to communicate with Oracle databases or third-party databases through JDBC. To access an existing relational schema, you use the Adapter Configuration Wizard to do the following:

- Import a relational schema and map it as an XML schema (XSD)
- Abstract SQL operations such as `SELECT`, `INSERT`, and `UPDATE` as Web services

While your BPEL process deals with XML and invokes web services, database rows and values are queried, inserted, and updated.

**See Also:** The following documentation for examples of using this adapter:

- SOA_Oracle_Home\bpel\samples\tutorials\122.DBAdapter

### Decision Service

This service enables the integration of a BPEL process with a rule set or function that is created in a business rules engine. With the decision service, you can make business decisions based on these rules. A wizard guides you through several steps to convert the selected rule set or function (and the operations to perform) into a Web service to use in the BPEL process. The wizard supports the Oracle Business Rules engine. A new decision service partner link for this Web service is automatically created that interfaces with the business rules engine.

Metadata information about the rules engine and repository is stored in a `.decs` configuration file in your BPEL project. This file is used during run time. A WSDL file based on the rule set to invoke is also created.
EJB Web Service
This service enables you to publish a Web service from a stateless session EJB. The wizard creates the WSDL document and deployment files needed to publish your code as a Web service.

See Also: The following documentation for examples of using this service:
- Chapter 18, "BPEL Process Integration with Business Rules"
- SOA_Oracle_Home\bpel\samples\demos\AutoLoanDemo

File Adapter
This adapter acts as both an inbound and outbound adapter. In the inbound direction, the adapter polls for files in a directory to retrieve and process. In the outbound direction, the adapter creates files in a directory.

See Also: The following documentation for examples of using this adapter:
- SOA_Oracle_Home\bpel\samples\tutorials\121.FileAdapter

FTP Adapter
This adapter acts as both an inbound and outbound adapter. In the inbound direction, the adapter polls for files in a directory to retrieve and process. In the outbound direction, the adapter creates files in a directory.

See Also: The following documentation for examples of using this adapter:
- SOA_Oracle_Home\bpel\samples\tutorials\129.FTPAdapter

Java Web Service
This service enables you to publish a Web service from a Java class. A wizard creates the WSDL document and deployment files needed to publish your code as a Web service. After you select the class and methods you want to publish, the wizard generates deployment descriptors, a JAX-RPC mapping file, and a WSDL document that can be deployed to an application server.

See Also: The following documentation for examples of using this adapter:
- SOA_Oracle_Home\bpel\samples\tutorials\123.JmsAdapter

JMS Adapter
This adapter acts as both a consume (inbound) and produce (outbound) messaging adapter. In the inbound direction, the adapter polls (consumes) messages from a JMS destination. In the outbound direction, the adapter sends (produces) messages to a JMS destination.

See Also: The following documentation for examples of using this adapter:
- SOA_Oracle_Home\bpel\samples\tutorials\123.JmsAdapter
MQ Adapter

This adapter provides message exchange capabilities between BPEL processes and the IBM MQSeries messaging software.

See Also: MQ adapter tutorials at
on/adapters/dev_support.html#tutorials

Oracle Applications

This adapter provides comprehensive, bidirectional, multimodal, synchronous, and asynchronous connectivity to Oracle Applications. The adapter supports all modules of Oracle Applications for versions 11.5.1 to 11.5.10. The adapter provides real-time and bidirectional connectivity to Oracle Applications through interface tables, views, application programming interfaces (APIs), and XML Gateway. The adapter inserts data into Oracle Applications using interface tables and APIs. To retrieve data from Oracle Applications, the adapter uses views. In addition, it uses XML Gateways for bidirectional integration with Oracle Applications. XML Gateways are also used to insert as well as receive Open Application Group Integration Specification (OAGIS)-compliant documents from Oracle Applications.

See Also: The following documentation for examples of using this adapter:
- Oracle Application Server Adapter for Oracle Applications User’s Guide
- SOA_Oracle_
  Home\bpel\samples\tutorials\150.AppsAdapter

PartnerLink

This service enables you to define the external services with which your process interacts. A partner link type characterizes the conversational relationship between two services by defining the roles played by each service in the conversation and specifying the port type provided by each service to receive messages within the context of the conversation. For example, if you are creating a process to interact with a Credit Rating Service and two loan provider services (United Loan and Star Loan), you create partner links for all three services.

When you double-click the PartnerLink icon, the Partner Link window shown in Figure B–30 appears. You provide the following details:
- A meaningful name for the service
- The Web services description language (WSDL) file of the external service
- The actual service type (defined as Partner Link Type)
- The role of the service (defined as Partner Role)
- The role of the process requesting the service (defined as My Role)
Validation When Loading a Process Diagram

**Figure B–30 PartnerLink Activity**

<table>
<thead>
<tr>
<th>General</th>
<th>Image</th>
<th>Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name:</td>
<td>CreditRating</td>
<td>PartnerLink:</td>
</tr>
<tr>
<td>Process:</td>
<td>OrderBooking</td>
<td>WSDL Settings</td>
</tr>
<tr>
<td>WSDL File:</td>
<td><a href="http://example.com/services/OrderBooking.wsdl">http://example.com/services/OrderBooking.wsdl</a></td>
<td></td>
</tr>
<tr>
<td>PartnerLink Type:</td>
<td>CreditRatingService</td>
<td>Partner Role:</td>
</tr>
<tr>
<td>Role:</td>
<td>CreditRatingServiceProvider</td>
<td></td>
</tr>
</tbody>
</table>

**See Also:** The following documentation for examples of using partner links:
- “Establishing the Partner Link” on page 4-2
- “partnerLinkTypes for Asynchronous Services” on page 5-3
- “Step 1: Adding a Partner Link for an Asynchronous Service” on page 5-11
- Oracle BPEL Process Manager Order Booking Tutorial
- Oracle BPEL Process Manager Quick Start Guide

Validation When Loading a Process Diagram

You may see an icon (a yellow triangle with an exclamation point) indicating invalid settings as you create and open activities such as a scope or an assign for the first time. The settings are invalid because you have not yet entered details.

To turn this option off for the current project, do the following:

1. Right-click the BPEL diagram and select **Display > Diagram Properties**.
2. Deselect the **Enable Automatic Validation** option.
3. Click **OK**.
4. Select **Save All** from the **File** main menu.

To disable this message for all new projects going forward, do the following:

1. Select **Preferences > BPEL Editor** from the **Tools** main menu in Oracle JDeveloper.
2. Deselect the **Enable Automatic Validation** option.
3. Click **OK**.
Changes Made In Oracle JDeveloper Do Not Update Automatically

Updates that you make in the BPEL Validation Browser window may not be automatically reflected in the BPEL process. For example, if you perform the following steps:

1. Create a partner link, do not specify a WSDL file, and click Apply and OK. This error causes an exclamation point icon to display in the upper left corner of the partnerLink icon.

2. Click the exclamation point to display the BPEL Validation Browser window.

3. Double-click the partner link in the BPEL Structure section of this window, correctly complete all fields in the Edit Partner Link window that displays, and apply and save your changes.

4. Right-click the BPEL Structure window and select Refresh Partner Links. Note that the partner link still displays an error.

As a workaround, close the BPEL Validation Browser window, double-click the partnerLink icon in the Diagram window of Oracle JDeveloper, and click Apply and OK. This action causes the error to disappear.

Summary

This appendix describes the process activities and services that you can drag and drop to create a BPEL process.
Deployment Descriptor Properties

This appendix discusses how to define deployment descriptor preference properties and deployment descriptor configuration properties.

This appendix contains the following topics:

- Deployment Descriptor Preference Properties
- Deployment Descriptor Configuration Properties
- Summary

Deployment Descriptor Preference Properties

Preferences are simple name-value pair properties that are defined in Oracle JDeveloper, and which are accessed at run time by the BPEL process. Preferences enable BPEL process designers to externalize literal values from a process. You can change the value of a preference at run time in Oracle BPEL Control, without having to redeploy the BPEL process.

For example, if you design a process that automatically rejects expense requests that exceed 1000 dollars, and business requirements later change so that the maximum amount is increased to 1500 dollars, then you normally need to edit the process definition and redeploy. By defining a preference for the maximum amount in the deployment descriptor property at design time, you can change the value at run time as needed, without redeploying the process.

Defining a Preference Property

You can define preference values in Oracle JDeveloper.

1. Click the Deployment Descriptor Properties icon, as shown in Figure C–1 (located in the upper-left area of a BPEL diagram).
2. In the Deployment Descriptor Properties window, click the Preferences tab, shown in Figure C–2.

3. Click Create and enter a preference name.

4. Edit the value in the Property Value field and click OK.

   The change takes effect immediately, and is reflected in bpe1.xml, in the preferences tag, as follows:
Updating a Preference at Run Time

You can update preference values at run time in Oracle BPEL Control.

1. Click the **BPEL Processes** tab.
2. Click a process name and then the **Descriptor** tab.

   The deployment descriptor for the process, including any preferences, is displayed, as shown in Figure C–3.

**Figure C–3 Updating BPEL Process Preferences at Run Time**

3. Update the preference value, and click **Update descriptor**.

   The change takes effect immediately.

Getting the Value of a Preference within a BPEL Process

The value of a preference can be read by a BPEL process using the XPath extension function `ora:getPreference(String preferenceName)`. This function can be used as part of a simple `assign` statement, used in condition expressions, or used as part of a more complex XPath expression.

Encrypting a Preference Value

You can encrypt the contents of a preference property. Encryption uses DES with the sunJCE security provider. The contents do not appear encrypted in Oracle JDeveloper. The contents are encrypted at deployment only. The **Encryption** list (see Figure C–2 on page C-2) provides the following options:

- **Plain Text**—The contents remain in plain text.
Encrypt on server on deploy—The contents remain in plain text in the Oracle JDeveloper project. However, the contents are encrypted on Oracle BPEL Server.

Encrypt on local machine—The contents remain in encrypted form in the Oracle JDeveloper project and are also encrypted on Oracle BPEL Server. The property is also shown as a password field in the Configurations tab of the deployment descriptor.

The following example shows the XML code without encryption set and then with encryption set.

Without encryption set:

```xml
...  
<preferences>
  <property name="secret">mySecretValue</property>
</preferences>
...
```

Or the XML without encryption can look as follows, although properties are stored as plaintext by default, so plaintext need not be specified explicitly.

```xml
...  
<preferences>
  <property name="secret" encryption="plaintext">mySecretValue</property>
</preferences>
...
```

To tell the compiler and Oracle BPEL Server to encrypt a property, the XML looks like this:

```xml
...  
<preferences>
  <property name="secret" encryption="encrypt">mySecretValue</property>
</preferences>
...
```

After the BPEL project is compiled, the compiler updates the copy of the bpel.xml file in the compiled JAR file (not the copy in the Oracle JDeveloper project), so that the XML looks like this:

```xml
...  
<preferences>
  <property name="secret" encryption="encrypted">ZAv9lfntAgy=</property>
</preferences>
...
```

Encryption works for any property tag in the descriptor, not just those in the preferences section, in case you want to encrypt properties in other sections.

**Note:** Values of preferences can still be inferred by inspecting the audit trails of instances that contain values derived from the preference.
are used to display a description of the process and default data in the test process window.

**Defining a Configuration Property**

You can define configuration properties in Oracle JDeveloper.

1. Click the **Deployment Descriptor Properties** icon, as shown in Figure C–1 on page C-2 (located in the upper-left area of a BPEL diagram).

2. In the Deployment Descriptor Properties window, click the **Configurations** tab, shown in Figure C–4.

3. Click **Create** and enter a configuration name.

4. Edit the value in the **Property Value** field and click **OK**.

   The change takes effect immediately, and is reflected in `bpel.xml`.

See "Encrypting a Preference Value" on page C-3 for information about encrypting the contents of configuration properties.

Table C–1 lists the property names of the `configurations` deployment descriptor.
<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
<th>BPEL Server Behavior on Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>completionPersistLevel</td>
<td>Sets the portion of the instance information that you want to save after the instance is completed. The default value is all, meaning the instance is saved in both the cube_instance and cube_scope tables. The other values are instanceHeader, meaning only the metadata of the instances are saved in the cube_instance table. Note that this property can only be set if the inMemoryOptimization property is set to True. See Also: Oracle Application Server Performance Guide for additional details about the inMemoryOptimization and completionPersistLevel properties.</td>
<td>NA</td>
</tr>
<tr>
<td>completionPersistPolicy</td>
<td>Configures how the instance data is saved. The default value is on, meaning the completed instance is saved normally. If this value is set to deferred, then the completed instance is saved, but with a different thread and in another transaction. If this value is set to faulted, then only the faulted instances are saved. If this value is set to off, then no instances of this process are saved. See Also: Oracle Application Server Performance Guide for additional details about the completionPersistPolicy property</td>
<td>NA</td>
</tr>
<tr>
<td>defaultInput</td>
<td>The XML document that you want to use as input to test the process from Oracle BPEL Control. Takes effect immediately</td>
<td></td>
</tr>
<tr>
<td>initializeVariables</td>
<td>Default value is True. If set to False, the compiler does not initialize the variables based on to-spec queries.</td>
<td>NA</td>
</tr>
<tr>
<td>inMemoryOptimization</td>
<td>Default value is False. This property can only be set to True if it does not have dehydration points. Activities like wait, receive, onMessage, and onAlarm create dehydration points in the process. If this property is set to True, Oracle BPEL Server tries to do in-memory optimization on the instances of this process on to-spec queries. See Also: Oracle Application Server Performance Guide for additional details about the inMemoryOptimization property</td>
<td>NA</td>
</tr>
<tr>
<td>loadSchema</td>
<td>Default value is True. If set to False, XML schemas are not loaded and Oracle BPEL Process Manager becomes typeless.</td>
<td>NA</td>
</tr>
<tr>
<td>noAlterWSDL</td>
<td>Default value is False. If set to True, the compiler does not try to modify the process WSDL to add binding and service information.</td>
<td>NA</td>
</tr>
<tr>
<td>optimizeVariableCopy</td>
<td>Default value is True. If set to False, Oracle BPEL Server does not enable copy-on-write for an assign copy.</td>
<td>NA</td>
</tr>
<tr>
<td>relaxTypeChecking</td>
<td>Default value is False. If set to True, the compiler does not check type compatibility with an assign activity.</td>
<td>NA</td>
</tr>
<tr>
<td>relaxXPathQName</td>
<td>Default value is False. If set to True, the compiler does not complain about unqualified steps in the query. For example, where the correct form must be: query=&quot;/ns1:payload/ns1:name&quot;, the following form passes compilation, if this flag is turned on: query=&quot;/payload/name&quot;.</td>
<td>NA</td>
</tr>
</tbody>
</table>
Table C–2 lists the configuration properties of sections of the partnerLinkBinding deployment descriptor.

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
<th>BPEL Server Behavior on Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>sensorActionLocation</td>
<td>Location of the sensor action XML file that is used by Oracle BPEL Process Manager. The sensor action XML file configures the action rule for the events.</td>
<td>NA</td>
</tr>
<tr>
<td>sensorLocation</td>
<td>Location of the sensor XML file. The sensor XML file defines the list of sensors into which Oracle BPEL Server logs events.</td>
<td>NA</td>
</tr>
<tr>
<td>testIntroduction</td>
<td>Introduction text that appears in the test console.</td>
<td>Takes effect immediately</td>
</tr>
<tr>
<td>transaction</td>
<td>When set to participate, the process produces a fault that is not handled by fault handlers, which calls the transaction to be rolled back.</td>
<td>Takes effect immediately</td>
</tr>
<tr>
<td>SLACompletionTime</td>
<td>Service Level Agreement (Completion Time) - Threshold for a commitment within which a process is completed for a specified time period. Value is an XML duration.</td>
<td>NA</td>
</tr>
<tr>
<td>xpathValidation</td>
<td>Default value is True. If set to False, the compiler does not validate the XPath queries.</td>
<td>NA</td>
</tr>
<tr>
<td>user</td>
<td>The username a calling user must provide (given the domain level security is on).</td>
<td>Redeploy the process</td>
</tr>
<tr>
<td>pw</td>
<td>The password a calling user must provide (given the domain level security is on).</td>
<td>Redeploy the process</td>
</tr>
<tr>
<td>role</td>
<td>The role a calling user must belong to in the identity management (given the domain level security is on).</td>
<td>Redeploy the process</td>
</tr>
</tbody>
</table>

Table C–2 lists the configuration properties of sections of the partnerLinkBinding deployment descriptor.

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
<th>BPEL Server Behavior on Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>basicHeaders</td>
<td>Creates HTTP basic authentication. The following values are supported:</td>
<td>Takes effect immediately</td>
</tr>
<tr>
<td></td>
<td>■ propagate — If the process has been invoked securely, these credentials are also used for the outbound direction</td>
<td></td>
</tr>
<tr>
<td></td>
<td>■ credentials — Passes credentials from the descriptor</td>
<td></td>
</tr>
<tr>
<td>basicUsername</td>
<td>The username (passed to basic authentication)</td>
<td>Takes effect immediately</td>
</tr>
<tr>
<td>basicPassword</td>
<td>The password credential (passed to basic authentication)</td>
<td>Takes effect immediately</td>
</tr>
<tr>
<td>callbackBindings</td>
<td>List of bindings that the compiler generates for the callback portType. The default value is soap. You set multiple bindings separated by commas (for example, jms, soap). The first item is used as the preferred binding when calling back.</td>
<td>Recompile (not implemented)</td>
</tr>
</tbody>
</table>
Table C–2 (Cont.) Configuration Properties for the partnerLinkBinding Deployment Descriptor

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
<th>BPEL Server Behavior on Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>correlation</td>
<td>Default value is wsAddressing. If this is set to correlationSet, this</td>
<td>If this is the process</td>
</tr>
<tr>
<td></td>
<td>partner link is using the BPEL</td>
<td>correlation set.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>recompile (not implemented)</td>
</tr>
<tr>
<td>contentType</td>
<td>Sets the special HTTP contentType. Example: text/xml</td>
<td>Takes effect immediately</td>
</tr>
<tr>
<td>httpAccept</td>
<td>Overwrites the HTTP accept header that Oracle BPEL Server sends to</td>
<td>Takes effect immediately</td>
</tr>
<tr>
<td></td>
<td>the remote SOAP service.</td>
<td></td>
</tr>
<tr>
<td>httpContentType</td>
<td>Overwrites the HTTP content-type header that Oracle</td>
<td>Takes effect immediately</td>
</tr>
<tr>
<td></td>
<td>BPEL Server sends to the remote SOAP service.</td>
<td></td>
</tr>
<tr>
<td>httpKeepAlive</td>
<td>If the server permits keepAlive connections, this Boolean property can</td>
<td>Takes effect immediately</td>
</tr>
<tr>
<td></td>
<td>be turned on to take advantage of it. Thus, connections to the same</td>
<td></td>
</tr>
<tr>
<td></td>
<td>server are shared between invocations.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>This attribute was previously</td>
</tr>
<tr>
<td>httpPassword</td>
<td>For HTTP username and password authentication</td>
<td>Takes effect immediately</td>
</tr>
<tr>
<td>httpUsername</td>
<td>For HTTP username and password authentication</td>
<td>Takes effect immediately</td>
</tr>
<tr>
<td>location</td>
<td>URL that overrides the location defined in the WSDL. For SOAP over HTTP</td>
<td>Takes effect immediately</td>
</tr>
<tr>
<td></td>
<td>binding, this value overrides the SOAP address.</td>
<td></td>
</tr>
<tr>
<td>nonBlockingInvoke</td>
<td>Default value is False. When this is set to True, Oracle BPEL Server</td>
<td>Takes effect immediately</td>
</tr>
<tr>
<td></td>
<td>spawns a separate thread to do the invocation so that the invoke activity</td>
<td></td>
</tr>
<tr>
<td></td>
<td>does not block the instance.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>See Also: Oracle Application Server Performance Guide for additional</td>
<td></td>
</tr>
<tr>
<td></td>
<td>details about the nonBlockingInvoke property</td>
<td></td>
</tr>
<tr>
<td>retryInterval</td>
<td>Number of seconds that Oracle BPEL Server waits between retries.</td>
<td>Takes effect immediately</td>
</tr>
<tr>
<td>retryMaxCount</td>
<td>Number of retries that Oracle BPEL Server attempts, if an invoke fails</td>
<td>Takes effect immediately</td>
</tr>
<tr>
<td></td>
<td>because of network problems.</td>
<td></td>
</tr>
<tr>
<td>sendXSIType</td>
<td>Some legacy RPC-style Web services require the xsi:type to be set with</td>
<td>Takes effect immediately</td>
</tr>
<tr>
<td></td>
<td>every element in the input message. If this value is set to True, Oracle</td>
<td></td>
</tr>
<tr>
<td></td>
<td>BPEL Process Manager populates the xsi:type of all the elements.</td>
<td></td>
</tr>
<tr>
<td>serviceProperties</td>
<td>--</td>
<td>Takes effect immediately</td>
</tr>
<tr>
<td>timeout</td>
<td>Number of seconds in which a SOAP call times out. A remote fault is</td>
<td>Takes effect immediately</td>
</tr>
<tr>
<td></td>
<td>thrown if this happens.</td>
<td></td>
</tr>
</tbody>
</table>

Deployment Descriptor Configuration Properties

C–8  Oracle BPEL Process Manager Developer’s Guide
This appendix discusses deployment descriptor preference properties and deployment descriptor configuration properties, and how to define them in Oracle JDeveloper.

### Table C–2 (Cont.) Configuration Properties for the partnerLinkBinding Deployment Descriptor

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
<th>BPEL Server Behavior on Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>validateXML</td>
<td>Enables message boundary validation. When set to true, Oracle BPEL Server validates the XML message against the XML schema during a receive activity and an invoke activity for this partner link. If the XML message is invalid, then a bpelex:invalidVariables runtime fault is thrown. This overrides the domain level validateXML property. The following example enables validation for only the StarLoanService partner:</td>
<td>Recompile</td>
</tr>
<tr>
<td></td>
<td>&lt;partnerLinkBinding name=&quot;StarLoanService&quot;&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;property name=&quot;wsdlLocation&quot;&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;property name=&quot;validateXML&quot;&gt;true&lt;/property&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;/partnerLinkBinding&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>See Also: Oracle Application Server Performance Guide for additional details about the validateXML property</td>
<td></td>
</tr>
<tr>
<td>wsdlLocation</td>
<td>URL of the WSDL file that defines this partner link. This property must be present. The BPEL compiler needs this to validate the BPEL source. This can be an abstract WSDL in that only the portTypes and their dependencies need to be defined in the WSDL.</td>
<td>Recompile (not implemented)</td>
</tr>
<tr>
<td>wsdlRuntimeLocation</td>
<td>URL to the partner link WSDL. It is used on Oracle BPEL Server, which means that the concrete WSDL with all the service, port, and binding definitions is needed. This property is optional and defaults to the wsdlLocation property. This property also enables multiple URLs separated by blanks (spaces, new lines, and tabs). Therefore, Oracle BPEL Server tries sequentially if any URLs are not available.</td>
<td>Clear WSDL cache (not implemented)</td>
</tr>
<tr>
<td>wsseHeaders</td>
<td>Creates a WS-Security username token. The following values are supported:</td>
<td>Takes effect immediately</td>
</tr>
<tr>
<td></td>
<td>■ propagate — If the process has been invoked securely, these credentials are also used for the outbound direction</td>
<td></td>
</tr>
<tr>
<td></td>
<td>■ credentials — Passes credentials from the descriptor</td>
<td></td>
</tr>
<tr>
<td></td>
<td>See Also: Oracle BPEL Process Manager Administrator's Guide for additional details about the wsseHeaders property</td>
<td></td>
</tr>
<tr>
<td>wsseUsername</td>
<td>The username for the token (required)</td>
<td>Takes effect immediately</td>
</tr>
<tr>
<td></td>
<td>See Also: Oracle BPEL Process Manager Administrator's Guide for additional details about the wsseUsername property</td>
<td></td>
</tr>
<tr>
<td>wssePassword</td>
<td>The password for the token (optional)</td>
<td>Takes effect immediately</td>
</tr>
<tr>
<td></td>
<td>See Also: Oracle BPEL Process Manager Administrator's Guide for additional details about the wssePassword property</td>
<td></td>
</tr>
</tbody>
</table>
XPath Extension Functions

Oracle provides additional XPath extension functions that use built-in BPEL capabilities and XPath standards.

This appendix contains the following topics:

- Advanced Functions
- BPEL Extension Functions
- BPEL XPath Extension Functions
- Database Functions
- Date Functions
- Mathematical Functions
- Identity Service Functions
- Workflow Service Functions
- String Functions
- Utility Functions
- Adding a Custom XPath Function
- Summary

See Also:
- "Deprecated Workflow Service and Identity Service Functions" on page 15-123 for information about workflow service and identity service functions that are deprecated for this release
- [http://www.w3.org](http://www.w3.org) for details about XPath functions

Advanced Functions

This section describes the following functions:

- create-nodeset-from-delimited-string
- generate-guid
- lookup-dvm
- lookup-xml
create-nodeset-from-delimited-string

The function takes a delimited string and returns a nodeSet.

Signature:

\texttt{orcl:create-nodeset-from-delimited-string(qname, delimited-string, delimiter)}

Arguments:

- \texttt{qname} - The qualified name in which each node in the node set must be created. The QName can be represented in two forms:
  - task:assignee
  - {http://mytask/task}assignee
- \texttt{delimited-string} - The sting of elements separated by the delimiter.
- \texttt{delimiter} - The character that separates the items in the input string; for example, a comma or a semicolon.

Property IDs:

- namespace-prefix: orcl

generate-guid

Generates a unique GUID.

Signature:

\texttt{orcl:generate-guid()}

Property IDs:

- namespace-prefix: orcl

lookup-dvm

This function returns a string by looking up the value for the target column in the domain value map (DVM), where the value for source column is equal to the source value. The source value is an XPath expression bound to the source document of the XSLT transformation. The expression is evaluated during the transformation and the result value is passed as the source value for lookup.

Note: Only use \texttt{lookup-dvm} with Oracle Enterprise Service Bus. Use of this function with Oracle BPEL Process Manager is not currently supported.
orcl:lookup-dvm(dvmName as string, sourceColumn as string, sourceValue as string, targetColumn as string, defaultValue as string)

Arguments:
- dvmName - The DVM name.
- sourceColumn - The source column name.
- sourceValue - The source value (an XPath expression bound to the source document of the XSLT transformation).
- targetColumn - The target column name.
- defaultValue - If the value is not found, the default value is returned.

Example:
You want to convert from abbreviated state names in the United States to their full names:

orcl:lookup-dvm('State Code', 'code', 'CA', 'state', 'Not Found')

This attempts to convert an abbreviated CA to its full name of California. The default value is Not Found.

Property IDs:

lookup-xml

This function returns the string value of an element defined by lookupXPath in an XML file (docURL) given its parent XPath (parentXPath), the key XPath (keyXPath), and the value of the key (key).

Example:
orcl:lookup-xml('file:/d:/country_data.xml', '/Countries/Country', 'Abbreviation', 'FullName', 'UK') returns the value of the element FullName child of /Countries/Country where Abbreviation = 'UK' is in the file D:\country_data.xml.

Signature:
orcl:lookup-xml(docURL, parentXPath, keyXPath, lookupXPath, key)

Arguments:
- docURL - The XML file
- parentXPath - The parent XPath
- keyXPath - The key XPath
- lookupXPath - The lookup XPath
- key - The key value

Property IDs:
BPEL Extension Functions

This section describes the following functions:

- **getLinkStatus**
- **getVariableData**
- **getVariableProperty**

### getLinkStatus

This function returns a Boolean value indicating the status of the link. If the status of the link is positive the value is `true`, otherwise the value is `false`. This function can only be used in a `join` condition.

The `linkName` argument refers to the name of an incoming link for the activity associated with the `join` condition.

**Signature:**

```
bpws:getLinkStatus ('linkName')
```

**Arguments:**

- `variableName` - The source variable for the data
- `propertyName` - The QName of the property

**Property IDs:**

- `namespace-prefix`: `bpws`

### getVariableData

This function extracts arbitrary values from BPEL variables.

When only the first argument is present, the function extracts the value of the variable, which in this case must be defined using an XML Schema simple type or element. Otherwise, the return value of this function is a node set containing the single node representing either an entire part of a message type (if the second argument is present and the third argument is absent) or the result of the selection based on the `locationPath` (if both optional arguments are present). If the given `locationPath` selects a node set of a size other than one during execution, the standard fault `bpws:selectionFailure` is thrown.

**Signature:**

```
bpws:getVariableData ('variableName', 'partName'?,'locationPath'?)
```

**Arguments:**

- `variableName` - The source variable for the data
- `partName` - The part to select from the variable (optional)
- `locationPath` - Provides an absolute location path (with `/` meaning the root of the document fragment representing the entire part) to identify the root of a subtree within the document fragment representing the part (optional).
**BPEL XPath Extension Functions**

**getProperty**
This function extracts arbitrary values from BPEL variables.
If the given property selects a node set of a size other than one during execution, the standard fault `bpws:selectionFailure` is thrown.

**Signature:**
`bpws:getVariableProperty ('variableName', 'propertyname')`

**Arguments:**
- `variableName` - The source variable for the data
- `propertyName` - The QName of the property
- `locationPath` - Provides an absolute location path (with / meaning the root of the document fragment representing the entire part) to identify the root of a subtree within the document fragment representing the part (optional).

**Property IDs:**
- namespace-prefix: `bpws`

**See Also:** The following documentation for examples of using this function:
- Chapter 3, “Manipulating XML Data in BPEL”
- “BPEL Code Example of the FlowN Activity” on page 6-6
- “Using a Switch Activity to Define Conditional Branching” on page 7-2
- “Setting E-mail Addresses and Telephone Numbers Dynamically” on page 14-13

**addQuotes**
This function returns the content of a string with single quotes added.

**Signature:**
`ora:addQuotes(string)`

**Arguments:**
- `string` - The string to which this function adds quotes

**Property IDs:**
**BPEL XPath Extension Functions**

- **namespace-uri**: http://schemas.oracle.com/xpath/extension
- **namespace-prefix**: ora

### appendToList

**Note:** While the `appendToList` function is still available for use, Oracle recommends that you use the `bpelx:append` extension activity of an `assign` activity to append data to a node list.

This function appends to a node list. The node list with which to append should not be null or empty.

**Signature:**

`ora:appendToList('variableName', 'partName'? , 'locationPath'? , Object)`

**Arguments:**

- **variableName** - The source variable for the data
- **partName** - The part to select from the variable (optional)
- **locationPath** - Provides an absolute location path (with `/` meaning the root of the document fragment representing the entire part) to identify the root of a subtree within the document fragment representing the part (optional).
- **Object** - The object can be either a list or a single item. If the object is a list, this function appends each item in the list. Each appended item is either an element, or an element with the string value of the node created.

**Property IDs:**

- **deprecated**
  
  Use the `bpelx:copyList` or `bpelx:append` extension activity to append to a list. This extension activity is demonstrated in sample SOA_Oracle_Home\bpel\samples\tutorials\126.DataAggregator.

- **namespace-uri**: http://schemas.oracle.com/xpath/extension
- **namespace-prefix**: ora

### copyList

**Note:** While the `copyList` function is still available for use, Oracle recommends that you use the `bpelx:copyList` extension to copy a node list or a node.

This function copies a node list or a node. The node list to be copied to should not be null or empty.

**Signature:**

`ora:copyList('variableName', 'partName'? , 'locationPath'? , Object)`

**Arguments:**
variableName - The source variable for the data

partName - The part to select from the variable (optional)

locationPath - Provides an absolute location path (with \ meaning the root of the document fragment representing the entire part) to identify the root of a subtree within the document fragment representing the part (optional)

Object - The object can be either a list or a single item. If the object is a list, each item in the list is copied. Each item to be copied is either an element, or an element with the string value of the node created.

Property IDs:

deprecated

Use the bpelx:copyList extension activity to append to a list. This extension activity is demonstrated in sample SOA_Oracle_Home\bpel\samples\tutorials\126.DataAggregator.

namespace-uri: http://schemas.oracle.com/xpath/extension

namespace-prefix: ora

countNodes

Signature:
ora:countNodes('variableName', 'partName'?, 'locationPath'?)

Arguments:
variableName - The source variable for the data

partName - The part to select from the variable (optional)

locationPath - Provides an absolute location path (with \ meaning the root of the document fragment representing the entire part) to identify the root of a subtree within the document fragment representing the part (optional)

Property IDs:
namespace-uri: http://schemas.oracle.com/xpath/extension
namespace-prefix: ora

doc

This function returns the content of an XML file.

Signature:
ora:doc('fileName','xpath'?)

Arguments:
fileName - The name of the XML file

Note: While the countNodes function is still available for use, Oracle recommends that you use version 1.0 of the XPath count() function to return the size of the elements as an integer.
**xpath**  
- The path to the file

**Property IDs:**
- namespace-uri:http://schemas.oracle.com/xpath/extension
- namespace-prefix:ora

### formatDate

This function converts standard XSD date formats to characters suitable for output.

**Signature:**
ora:formatDate('dateTime', 'format')

**Arguments:**
- `dateTime` - Contains a date-related value in XSD format. For nonstring arguments, this function behaves as if a `string()` function were applied. If the argument is not a date, the output is an empty string. If it is a valid XSD date and some fields are empty, this function attempts to fill unspecified fields. For example, 2003-06-10T15:56:00.
- `format` - Contains a string formatted according to `java.text.SimpleDateFormat` format

**Property IDs:**
- namespace-uri:http://schemas.oracle.com/xpath/extension
- namespace-prefix:ora

**See Also:** "Assigning Date or Time" on page 3-8 for an example of using this function

### generateGUID

Generates a unique GUID.

**Signature:**
ora:generateGUID()

**Property IDs:**
- namespace-uri:http://schemas.oracle.com/xpath/extension
- namespace-prefix:ora

### getContentAsString

This function returns the content of an element as an XML string.

**Signature:**
ora:getContentAsString(NodeList elementAsNodeList)

**Arguments:**
- `NodeList` - The node list (source for the data).
- `elementAsNodeList` - The element as the node list.

**Property IDs:**
- namespace-uri:http://schemas.oracle.com/xpath/extension
getConversationId

This function returns the conversation ID.

Signature:
ora:getConversationId()

Property IDs:
- namespace-uri: http://schemas.oracle.com/xpath/extension
- namespace-prefix: ora

getCreator

This function returns the instance creator.

Signature:
ora:getCreator()

Property IDs:
- namespace-uri: http://schemas.oracle.com/xpath/extension
- namespace-prefix: ora

getCurrentDate

This function returns the current date as a string.

Signature:
ora:getCurrentDate('format'?)

Argument:
- format -(Optional) Specifies a string formatted according to java.text.SimpleDateFormat format (optional).

Property IDs:
- namespace-uri: http://schemas.oracle.com/xpath/extension
- namespace-prefix: ora

See Also: "Assigning Date or Time" on page 3-8 for an example of using this function

getCurrentDateTime

This function returns the current date time as a string.

Signature:
ora:getCurrentDateTime('format'?)

Argument:
- format -(Optional) Specifies a string formatted according to java.text.SimpleDateFormat format (optional).

Property IDs:
BPEL XPath Extension Functions

- namespace-uri: http://schemas.oracle.com/xpath/extension
- namespace-prefix: ora

**getCurrentTime**
This function returns the current time as a string.

**Signature:**
ora:getCurrentTime('format'?)

**Argument:**
- format - (Optional) Specifies a string formatted according to java.text.SimpleDateFormat format (optional).

**Property IDs:**
- namespace-uri: http://schemas.oracle.com/xpath/extension
- namespace-prefix: ora

**getDomainId**
This function returns the current domain ID.

**Signature:**
ora:getDomainId()

**Property IDs:**
- namespace-uri: http://schemas.oracle.com/xpath/extension
- namespace-prefix: ora

**getElement**
This function returns an element using index from the array of elements.

**Signature:**
ora:getElement('variableName', 'partName', 'locationPath', index)

**Arguments:**
- variableName - The source variable for the data
- partName - The part to select from the variable (required)
- locationPath - Provides an absolute location path (with / meaning the root of the document fragment representing the entire part) to identify the root of a subtree within the document fragment representing the part (required)
- index - Dynamic index value. The index of the first node is 1.

**Property IDs:**
- namespace-uri: http://schemas.oracle.com/xpath/extension
- namespace-prefix: ora

See Also: "Dynamically Indexing with the BPEL getElement Function" on page 3-19 for an example of using this function
getGroupIdsFromGroupAlias

This function returns a list of user IDs for a group alias specified in the TaskServiceAliases section of the BPEL suitcase descriptor.

Signature:
ora:getGroupIdsFromGroupAlias(String aliasName)

Arguments:
- aliasName - The alias for a list of users or groups as defined in the bpel.xml file

Property IDs:
- namespace-uri: http://schemas.oracle.com/xpath/extension
- namespace-prefix: ora

getInstanceId

This function returns the instance ID.

Signature:
ora:getInstanceId()

Property IDs:
- namespace-uri: http://schemas.oracle.com/xpath/extension
- namespace-prefix: ora

getNodeValue

This function returns the value of a DOM node as a string.

Signature:
ora:getNodeValue(node)

Arguments:
- node - The DOM node

Property IDs:
- namespace-uri: http://schemas.oracle.com/xpath/extension
- namespace-prefix: ora

getNodes

This function gets a node list. This is implemented as an alternate to bpws:getVariableData, which does not return a node list.

Signature:
ora:getNodes('variableName', 'partName'? , 'locationPath'? )

Arguments:
- variableName - The source variable for the data
- partName - The part to select from the variable (optional)
BPEL XPath Extension Functions

- **locationPath** - Provides an absolute location path (with / meaning the root of the document fragment representing the entire part) to identify the root of a subtree within the document fragment representing the part (optional).

  Property IDs:
  - namespace-uri: http://schemas.oracle.com/xpath/extension
  - namespace-prefix: ora

  **See Also:** "getVariableData" on page D-4

**getPreference**

This function returns the value of a property specified in the preferences section of the BPEL suitcase descriptor.

**Signature:**
ora:getPreference(String preferenceName)

**Arguments:**
- **preferenceName** - The name of the preference as specified in the BPEL suitcase descriptor.

  Property IDs:
  - namespace-uri: http://schemas.oracle.com/xpath/extension
  - namespace-prefix: ora

**getProcessId**

This function returns the ID of the current BPEL process.

**Signature:**
ora:getProcessId()

Property IDs:
- namespace-uri: http://schemas.oracle.com/xpath/extension
- namespace-prefix: ora

**getProcessOwnerId**

This function returns the ID of the user who owns the process, if specified in the TaskServiceAliases section of the BPEL suitcase descriptor.

**Signature:**
ora:getProcessOwnerId()

Property IDs:
- namespace-uri: http://schemas.oracle.com/xpath/extension
- namespace-prefix: ora

**getProcessURL**

This function returns the root URL of the current BPEL process.

**Signature:**

D-12 Oracle BPEL Process Manager Developer’s Guide
ora:getProcessURL()

Property IDs:
- namespace-uri: http://schemas.oracle.com/xpath/extension
- namespace-prefix: ora

getProcessVersion

This function returns the current process version.

Signature:
ora:getProcessVersion()

Property IDs:
- namespace-uri: http://schemas.oracle.com/xpath/extension
- namespace-prefix: ora

getUserAliasId

This function returns the user ID for an alias specified in the TaskServiceAliases section of the BPEL suitcase descriptor.

Signature:
ora:getUserAliasId (String aliasName)

Arguments:
- aliasName - The alias for a list of users or groups as defined in the bpel.xml file.

Property IDs:
- namespace-uri: http://schemas.oracle.com/xpath/extension
- namespace-prefix: ora

integer

This function returns the content of the node as an integer.

Signature:
ora:integer(node)

Arguments:
- node - The input node

Property IDs:
- namespace-uri: http://schemas.oracle.com/xpath/extension
- namespace-prefix: ora

parseEscapedXML

This function parses a string to DOM.

Signature:
ora:parseEscapedXML(contentString)
BPEL XPath Extension Functions

Arguments:
- contentString - The string that this function parses to a DOM.

Property IDs:
- namespace-uri:http://schemas.oracle.com/xpath/extension
- namespace-prefix:ora

See Also:  "Converting from a String to an XML Element" on page 3-21 for an example of using this function

**processXQuery**

This function returns the result of an XQuery transformation.

Signature:
ora:processXQuery('template','context'?)

Arguments:
- template - The XSLT template
- input - The input data to be transformed
- properties - The properties as defined in the bpel.xml file

Property IDs:
- namespace-uri:http://schemas.oracle.com/xpath/extension
- namespace-prefix:ora

**processXSLT**

This function returns the result of XSLT transformation using the Oracle XDK XSLT processor.

Signature:
xdk:processXSLT('template','input','properties'?)

Arguments:
- template - The XSLT template
- input - The input data to be transformed
- properties - The properties as defined in the bpel.xml file

Property IDs:
- namespace-uri:http://schemas.oracle.com/xpath/extension
- namespace-prefix:xdk

See Also:
- "114.XSLTTransformations" on page 1-11 for a tutorial that uses this function
- "Formatting the Body of an E-mail Message as HTML" on page 14-8 for an example of using this function
processXSQL

This function returns the result of the XSQL request.

Signature:
ora:processXSQL('template','input','properties'?)

Arguments:
- template - The XSLT template
- input - The input data to be transformed
- properties - The properties as defined in the bpe1.xml file

Property IDs:
- namespace-uri: http://schemas.oracle.com/xpath/extension
- namespace-prefix: ora

readBinaryFromFile

This function reads data from a file.

Signature:
ora:readBinaryFromFile(fileName)

Arguments:
- fileName - The file name from which to read data.

Property IDs:
- namespace-uri: http://schemas.oracle.com/xpath/extension
- namespace-prefix: ora

See Also: "BPEL File Contents" on page 3-32

readFile

This function returns the content of the file.

Signature:
ora:readFile('fileName','nxsdTemplate'?,'nxsdRoot'?)

Arguments:
- fileName - The name of the file. This argument can also be an HTTP URL.

This function by default reads files relative to the suitcase JAR file for the process. If the file that you want to read is located in a different directory path, you must specify an extra directory slash (/) to indicate that this is an absolute path. For example:
ora:readFile('file:///c:/temp/test.doc')

If you specify only two directory slashes (//), you receive an error similar to the following:
XPath expression failed to execute.
Error while processing xpath expression, the expression is "ora:readFile('file:///c://temp/test.doc')", the reason is c. Please verify the xpath query.
Database Functions

- nxsdTemplate - The NXSD template for the output
- nxsdRoot - The NXSD root

Property IDs:
- namespace-uri: http://schemas.oracle.com/xpath/extension
- namespace-prefix: ora

See Also: "Setting E-mail Attachments" on page 14-5 for an example of using this function

writeBinaryToFile

This function writes the binary bytes of a variable (or part of the variable) to a file of the given file name.

Signature:
ora:writeBinaryToFile(varName[, partName][, query])

Arguments:
- varName - The name of the variable.
- partName - The name of the part in the case of a messageType variable.
- query - The query string to a child of the root element.

Property IDs:
- namespace-uri: http://schemas.oracle.com/xpath/extension
- namespace-prefix: ora

See Also: "BPEL File Contents" on page 3-32

Database Functions

This section describes the following functions:
- lookup-table
- query-database
- sequence-next-val

lookup-table

This function returns a string based on the SQL query generated from the parameters.

The string is obtained by executing:
SELECT outputColumn FROM table WHERE inputColumn = key

against the data source that can be either a JDBC connect string (jdbc:oracle:thin:username/password@host:port:sid) or a data source JNDI identifier. Only Oracle Thin Driver is supported if the JDBC connect string is used.


See Also: "BPEL File Contents" on page 3-32
Signature:

orcl:lookup-table(table, inputColumn, key, outputColumn, datasource)

Arguments:

- **table** - The table from which to draw the data
- **inputColumn** - The column within the table
- **key** - The key
- **outputColumn** - The column to output the data
- **datasource** - The source of the data

Property IDs:

- namespace-prefix: orcl

---

**query-database**

This function returns a node-set by executing the SQL query against the specified database.

Signature:

orcl:query-database(sqlquery as string, rowset as boolean, row as boolean, datasource as string)

Arguments:

- **sqlquery** - The SQL query to perform
- **rowset** - Indicates if the rows should be enclosed in an element
- **row** - Indicates if each row should be enclosed in an element
- **datasource** - Either a JDBC connect string 
  (jdbc:oracle:thin:username/password@host:port:sid) or a JNDI name for the database

Property IDs:

- namespace-prefix: orcl

---

**sequence-next-val**

Returns the next value of an Oracle sequence.

The next value is obtained by executing

```sql
SELECT sequence.nextval FROM dual
```

against a data source that can be either a JDBC connect string 
jdbc:oracle:thin:username/password@host:port:sid or a data source
JNDI identifier. Only Oracle Thin Driver is supported if a JDBC connect string is used.
Date Functions

Example: `orcl:sequence-next-val('employee_id_sequence','jdbc:oracle:thin:scott/tiger@localhost:1521:ORCL')`

Signature:
`orcl:sequence-next-val(sequence as string, datasource as string)`

Arguments:
- `sequence` - The sequence number in the database
- `datasource` - Either a JDBC connect string or a data source JNDI identifier

Property IDs:
- `namespace-prefix`: orcl

Date Functions

This section describes the following functions:

**add-dayTimeDuration-to-dateTime**

This function returns a new date time value adding `dateTime` to the given duration. If the duration value is negative, then the resulting value precedes `dateTime`.

Signature:
`xp20:add-dayTimeDuration-from-dateTime(dateTime as string, duration as string)`

Arguments:
- `dateTime as string` - The `dateTime` to which the function adds the duration, in string format.
- `duration as string` - The duration to add to the `dateTime`, or subtract if the duration is negative, in string format.

Property IDs:
- `namespace-prefix`: xp20

**current-date**

This function returns the current date in ISO format YYYY-MM-DD.

Signature:
`xp20:current-date(object)`

Arguments:
- `Object` - The time in standard format

Property IDs:
current-dateTime

This function returns the current datetime-value in ISO format CCYY-MM-DDThh:mm:ssTZD.

Signature:
xp20:current-dateTime(object)

Arguments:
■ object - The time in standard format

Property IDs:
■ namespace-uri:
  http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.Xpath20
■ namespace-prefix: xp20

See Also: “Setting Dynamic Values at Run Time” on page 20-27 for an example of using this function

current-time

This function returns the current time in ISO format. The format is hh:mm:ssTZD.

Signature:
xp20:current-time(object)

Arguments:
■ object - The time in standard format

Property IDs:
■ namespace-uri:
  http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.Xpath20
■ namespace-prefix: xp20

day-from-dateTime

This function returns the day from dateTime. The default day is 1.

Signature:
xp20:day-from-dateTime(object)

Arguments:
■ object - The time in standard format as a string

Property IDs:
Date Functions

- **namespace-uri**: http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.Xpath20
- **namespace-prefix**: xp20

**format-dateTime**

This function returns the formatted string of dateTime using the format provided.

**Signature:**

xp20:format-dateTime(dateTime as string, format as string)

**Arguments:**

- **dateTime** - The dateTime to be formatted
- **format** - The format for the output

**Property IDs:**

- **namespace-uri**: http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.Xpath20
- **namespace-prefix**: xp20

**hours-from-dateTime**

This function returns the hour from dateTime. The default hour is 0.

**Signature:**

xp20:hours-from-dateTime(dateTime as string)

**Arguments:**

- **dateTime as string** - The dateTime

**Property IDs:**

- **namespace-uri**: http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.Xpath20
- **namespace-prefix**: xp20

**implicit-timezone**

This function returns the current time zone in ISO format +/- hh:mm, indicating a deviation from UTC (Coordinated Universal Timezone).

**Signature:**

xp20:implicit-timezone(object)

**Arguments:**

- **object** - The time in standard format

**Property IDs:**

- **namespace-uri**: http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.Xpath20
Date Functions

XPath Extension Functions

- namespace-prefix: xp20

**minutes-from-dateTime**

This function returns the minute from `dateTime`. The default minute is 0.

Signature:

```
xp20:minutes-from-dateTime(dateTime as string)
```

Arguments:
- `dateTime as string` - The `dateTime`

Property IDs:
- namespace-prefix: xp20

**month-from-dateTime**

This function returns the month from `dateTime`. The default month is 1 (January).

Signature:

```
xp20:month-from-dateTime(dateTime as string)
```

Arguments:
- `dateTime as string` - The `dateTime` to be formatted

Property IDs:
- namespace-prefix: xp20

**seconds-from-dateTime**

This function returns the second from `dateTime`. The default second is 0.

Signature:

```
xp20:seconds-from-dateTime(dateTime as string)
```

Arguments:
- `dateTime as a string` - The `dateTime` as a string

Property IDs:
- namespace-prefix: xp20

**subtract-dayTimeDuration-from-dateTime**

This function returns a new `dateTime` value after subtracting duration from `dateTime`.

XPath Extension Functions  D-21
If the duration value is negative, then the resultant dateTime value follows input-datetime value.

Signature:
```
xp20:subtract-dayTimeDuration-from-dateTime(dateTime as string, duration as string)
```

Arguments:
- **dateTime as string**: The dateTime from which the function subtracts the duration, in string format.
- **duration as string**: The duration to subtract to the dateTime, or add if the duration is negative, in string format.

Property IDs:
- **namespace-uri**: `http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.Xpath20`
- **namespace-prefix**: `xp20`

**timezone-from-dateTime**

This function returns the timezone from dateTime. The default timezone is GMT+00:00.

Signature:
```
xp20:timezone-from-dateTime(dateTime as string)
```

Arguments:
- **dateTime as string**: The dateTime for which this function returns a timezone.

Property IDs:
- **namespace-uri**: `http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.Xpath20`
- **namespace-prefix**: `xp20`

**year-from-dateTime**

This function returns the year from dateTime.

Signature:
```
xp20:year-from-dateTime(dateTime as string)
```

Arguments:
- **dateTime**: The dateTime

Property IDs:
- **namespace-uri**: `http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.Xpath20`
- **namespace-prefix**: `xp20`
Mathematical Functions

This section describes the following function.

abs

This function returns the absolute value of inputNumber.

If inputNumber is not negative, the inputNumber is returned. If the inputNumber is negative, the negation of inputNumber is returned.

Example: abs(-1) returns 1.

Signature:

gp20:abs(inputNumber as number)

Arguments:

- inputNumber as number - The number for which the function returns an absolute value.

Property IDs:

- namespace-uri:
  http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.Xpath20
- namespace-prefix: gp20

Identity Service Functions

This section describes the following functions:

- getDefaultRealmName
- getGroupProperty
- getManager
- getReportees
- getSupportedRealmNames
- getUserProperty
- getUserRoles
- getUsersInGroup
- isUserInRole
- lookupGroup
- lookupUser

getDefaultRealmName

This function returns the default realm name.

Signature:

gp20:getDefaultRealmName()
Identity Service Functions

- namespace-uri: http://xmlns.oracle.com/bpel/services/IdentityService/xpath
- namespace-prefix: ids

**getGroupProperty**

This function returns the property value for the given group. If the group or attribute does not exist, it returns `null`.

**Signature:**

`ids:getGroupProperty(groupName, attributeName, realmName)`

**Arguments:**

- **groupName** - The group name.
- **attributeName** - The name of the group attribute. The name is one of the following values:
  1. name
  2. displayName
  3. description
  4. email

If the identity service uses the LDAP `providerType` or JAZN LDAP-based providers, configure the LDAP server to enable searching by those attributes.

- **realmName** - The realm name. This is optional. If not specified, the default realm is assumed.

**Property IDs:**

- namespace-uri: http://xmlns.oracle.com/bpel/services/IdentityService/xpath
- namespace-prefix: ids

**getManager**

This function gets the manager of a given user. If the user does not exist or there is no manager for this user, it returns `null`.

**Signature:**

`ids:getManager(userName, realmName)`

**Arguments:**

- **userName** - The user name.
- **realmName** - The realm name. This is optional. If not specified, the default realm is assumed.

**Property IDs:**

- namespace-uri: http://xmlns.oracle.com/bpel/services/IdentityService/xpath
- namespace-prefix: ids
getReportees
This function gets the reportees of the user. If the user does not exist, it returns null.
The function returns a list of nodes. Each node in the list is a user.
Signature:
ids:getReportees(userName, upToLevel, realmName)
Arguments:
- userName - The user name.
- upToLevel - Defines the levels of indirect reportees to be included into the result.
  If the value is 1, it returns only direct reportees. If the value is -1, it returns all
  levels of reportees. It can be either an element with value xsd:number or a string,
  for example: '1'.
- realmName - The realm name. This is optional and if not specified, the default
  realm is assumed.
Property IDs:
- namespace-uri: http://xmlns.oracle.com/bpel/services/IdentityService(xpath
- namespace-prefix: ids

getSupportedRealmNames
This function returns the supported realm names.
Signature:
ids:getSupportedRealms()
Property IDs:
- namespace-uri: http://xmlns.oracle.com/bpel/services/IdentityService(xpath
- namespace-prefix: ids

getUserProperty
This function returns the property of the user. If the user or attribute does not exist, it
returns null.
Signature:
ids:getUserProperty(userName, attributeName, realmName)
Arguments:
- userName - The user name.
- attributeName - The name of the user attribute. The attribute name is one of the
  following values:
  1. name
  2. displayName
  3. description
  4. title
  5. firstName
Identity Service Functions

6. middleName
7. lastName
8. workPhone
9. homePhone
10. mobile
11. pager
12. fax

If the identity service uses the LDAP providerType or JAZN LDAP-based providers, configure the LDAP server to enable searching by those attributes.

- realmName - The realm name. This is optional. If not specified, the default realm name is assumed

Property IDs:
- namespace-uri: http://xmlns.oracle.com/bpel/services/IdentityService/xpath
- namespace-prefix: ids

See Also: "Setting E-mail Addresses and Telephone Numbers Dynamically" on page 14-13 for an example

getUserRoles
This function gets the user roles. This function returns a list of objects, either application roles or groups, depending on the roleType. If the user or role does not exist, it returns null.

Signature:
ids:getUserRoles(userName, roleType, direct)

Arguments:
- userName - The user name.
- roleType - The role type — this is one of three values: ApplicationRole, EnterpriseRole, or AnyRole.
- direct - A Boolean flag indicating if direct or indirect roles are fetched. This is optional. If not specified, only direct roles are fetched. It can be either an element with value xsd:boolean or string 'true'/'false'.

Property IDs:
- namespace-uri: http://xmlns.oracle.com/bpel/services/IdentityService
- namespace-prefix: ids

getUsersInGroup
This function gets the users in a group. If the group does not exist, it returns null. The function returns a list of nodes. Each node in the list is called user.

Signature:
ids:getUsersInGroup(groupName, direct, realmName)
Arguments:
- **groupName** - The group name.
- **direct** - A Boolean flag. If **true**, the function returns direct user grantees; otherwise, all user grantees are returned. It can be either an element with value **xsd:boolean** or string "true"/'false'.
- **realmName** - The realm name. This is optional. If not specified, the default realm name is assumed.

Property IDs:
- **namespace-uri**: http://xmlns.oracle.com/bpel/services/IdentityService/xpath
- **namespace-prefix**: ids

**isUserInRole**

This function verifies if a user has a given role, returning a Boolean **true** or **false**. If the user does not exist, it returns **null**.

**Signature:**
ids:isUserInRole(userID, roleName, realmName)

**Arguments:**
- **userID** - The user name.
- **roleName** - The role or group name
- **realmName** - The realm name. This is optional. If not specified, the default realm name is assumed.

Property IDs:
- **namespace-uri**: http://xmlns.oracle.com/bpel/services/IdentityService/xpath
- **namespace-prefix**: ids

**lookupGroup**

This function gets the group. If the group does not exist, it returns **null**.

**Signature:**
ids:lookupGroup(groupName, realmName)

**Arguments:**
- **groupName** - The group name.
- **realmName** - The realm name. This is optional. If not specified, the default realm name is assumed.

Property IDs:
- **namespace-uri**: http://xmlns.oracle.com/bpel/services/IdentityService/xpath
- **namespace-prefix**: ids
lookupUser

This function gets the user object. If the user does not exist, it returns null.

Signature:
ids:lookupUser(userName, realmName)

Arguments:
- **userName** - The user name.
- **realmName** - The realm name. This is optional. If not specified, the default realm name is assumed.

Property IDs:
- namespace-uri: http://xmlns.oracle.com/bpel/services/IdentityService/xpath
- namespace-prefix: ids

Workflow Service Functions

This section describes the following functions:
- **clearTaskAssignees**
- **createWordMLDocument**
- **getNotificationProperty**
- **getNumberOfTaskApprovals**
- **getPreviousTaskApprover**
- **getTaskAttachmentByIndex**
- **getTaskAttachmentByName**
- **getTaskAttachmentContents**
- **getTaskAttachmentsCount**
- **getTaskResourceBundleString**
- **wfDynamicGroupAssign**
- **wfDynamicUserAssign**

**clearTaskAssignees**

This function clears the current task assignees.

Signature:
hwf:clearTaskAssignees(taskID)

Arguments:
- **task** - The task ID of the task.

Property IDs:
- namespace-uri: http://xmlns.oracle.com/bpel/workflow/xpath
- namespace-prefix: hwf
createWordMLDocument
This function creates a Microsoft Word ML document as a base 64-encoded string.

Signature:
hwf:createWordMLDocument(node, xsltURI)

Arguments:
■ node - The node
■ xsltURI - The XSLT used to transform the node (the first argument) to Microsoft Word ML

Property IDs:
■ namespace-uri: http://xmlns.oracle.com/bpel/workflow/xpath
■ namespace-prefix: hwf

See Also: “Specifying WordML Style Sheets for Attachments” on page 15-48

getNotificationProperty
This function retrieves a notification property. The function evaluates to corresponding values for each notification. Only use this function in the notification content XPath expression. If used elsewhere, it returns null.

Signature:
hwf:getNotificationProperty(propertyName)

Arguments:
■ propertyName - The name of the notification property. It can be one of the following values:
  = recipient - The recipient of the notification.
  = recipientDisplay - The display name of the recipient.
  = taskAssignees - The task assignees.
  = taskAssigneesDisplay - The display names of the task assignees.
  = locale - The locale of the recipient.
  = taskId - The task ID of the task for which the notification is meant.
  = taskNumber - The task number of the task for which the notification is meant.
  = appLink - The HTML link to the Oracle BPEL Worklist Application task details page.

Property IDs:
■ namespace-uri: http://xmlns.oracle.com/bpel/workflow/xpath
■ namespace-prefix: hwf

See Also:
■ “Contents of Notification” on page 15-82 for an example of using this function
■ “Configuring Messages in Different Languages” on page 15-83
Workflow Service Functions

getNumberOfTaskApprovals

This function computes the number of times the task was approved.

Signature:

```
hwf:getNumberOfTaskApprovals(taskId)
```

Arguments:

- `taskId` - The ID of the task

Property IDs:

- `namespace-uri`: `http://xmlns.oracle.com/bpel/workflow/xpath`
- `namespace-prefix`: `hwf`

getPreviousTaskApprover

This function retrieves the previous task approver.

Signature:

```
hwf:getPreviousTaskApprover(taskId)
```

Arguments:

- `taskId` - The ID of the task

Property IDs:

- `namespace-uri`: `http://xmlns.oracle.com/bpel/workflow/xpath`
- `namespace-prefix`: `hwf`

getTaskAttachmentByIndex

This function retrieves the task attachment at the specified index.

Signature:

```
hwf:getTaskAttachmentByIndex(taskId, attachmentIndex)
```

Arguments:

- `taskId` - The task ID of the task
- `attachmentIndex` - The index of the attachment. The index begins from 1. The `attachmentIndex` argument can be a node whose value evaluates to the index number as a string (all node values are strings). If specified statically, it can be specified as `'1'`.

Property IDs:

- `namespace-uri`: `http://xmlns.oracle.com/bpel/workflow/xpath`
- `namespace-prefix`: `hwf`

getTaskAttachmentByName

This function retrieves the task attachment by the attachment name.

Signature:

```
hwf:getTaskAttachmentByName(taskId, attachmentName)
```

Arguments:

D-30 Oracle BPEL Process Manager Developer's Guide
**Workflow Service Functions**

- `taskId` - The task ID of the task.
- `attachmentName` - The name of the attachment.

**Property IDs**:
- `namespace-uri`: http://xmlns.oracle.com/bpel/workflow/xpath
- `namespace-prefix`: hwf

**getTaskAttachmentContents**
This function retrieves the task attachment contents by the attachment name.

**Signature**:
```
hwf:getTaskAttachmentContents(taskId, attachmentName)
```

**Arguments**:
- `taskId` - The task ID of the task.
- `attachmentName` - The name of the attachment.

**Property IDs**:
- `namespace-uri`: http://xmlns.oracle.com/bpel/workflow/xpath
- `namespace-prefix`: hwf

**getTaskAttachmentsCount**
This function retrieves the number of task attachments.

**Signature**:
```
hwf:getTaskAttachmentsCount(taskId)
```

**Arguments**:
- `taskId` - The task ID.

**Property IDs**:
- `namespace-uri`: http://xmlns.oracle.com/bpel/workflow/xpath
- `namespace-prefix`: hwf

**getTaskResourceBundleString**
This function returns the internationalized resource value from the resource bundle associated with a task definition.

**Signature**:
```
hwf:getTaskResourceBundleString(taskId, key, locale?)
```

**Arguments**:
- `taskId` - The task ID of the task.
- `key` - The key to the resource.
- `locale` - (Optional) The locale. This value defaults to system locale. This returns a `resourceString` XML element in the namespace http://xmlns.oracle.com/bpel/services/taskService, which contains the string from the resource bundle.
Workflow Service Functions

Property IDs:
- namespace-uri: http://xmlns.oracle.com/bpel/workflow/xpath
- namespace-prefix: hwf

See Also: "Specifying Multilingual Settings" on page 15-48 and "Configuring Messages in Different Languages" on page 15-83 for additional details about this function

wfDynamicGroupAssign

This function gets the name of an identity service group, selected according to the specified assignment pattern. The group is selected from either the subordinate groups of the specified group (if a single group name is supplied), or from the list of groups (if a list of user names is supplied). If the identity service is configured with multiple realms, the realm name for the group and groups must also be supplied. Additional assignment pattern specific parameters can be supplied. These additional parameters are optional, depending on the details of the specific assignment pattern used.

There are two signatures of this function.

Signature 1:

```
hwf:wfDynamicGroupAssign('patternName','groupName','realmName'?, 'patternParam1'?,'patternParam2'?,...,'patternParamN'?)
```

Argument 1:
- `patternName` - The name of the assignment pattern (for example, ROUND_ROBIN)
- `groupName` - The name of the group from which to select a subordinate group.
- `realmName` - The name of the identity service realm to which the group belongs.
- `patternParam1...patternParamN` - Any additional parameters required by the assignment pattern implementation (may be optional, depending on pattern).

Signature 2:

```
hwf:wfDynamicGroupAssign('patternName','groupList','realmName'?, 'patternParam1'?,'patternParam2'?,...,'patternParamN'?)
```

Argument 2:
- `patternName` - The name of the assignment pattern (for example, ROUND_ROBIN)
- `groupList` - The list of groups from which to select a group.
- `realmName` - The name of the identity service realm to which the groups belong.
- `patternParam1...patternParamN` - Any additional parameters required by the assignment pattern implementation (may be optional, depending on the pattern).

Property IDs:
- namespace-uri: http://xmlns.oracle.com/bpel/workflow/xpath
- namespace-prefix: hwf

See Also: "Dynamic Assignment Functions" on page 15-113
**wfDynamicUserAssign**

This function returns the name of an identity service user, selected according to the specified assignment pattern. The user is selected from either the subordinate users of the specified group (if a single group name is supplied), or from the list of users (if a list of user names is supplied). If the identity service is configured with multiple realms, the realm name for the group and users must also be supplied. Additional assignment pattern specific parameters can be supplied. These additional parameters are optional, depending on the details of the specific assignment pattern used.

There are two signatures for this function.

**Signature 1:**

```java
hwf:wfDynamicUserAssign('patternName','groupName','realmName?',
  'patternParam1'?,...,'patternParam2'?,...,'patternParamN'?)
```

**Arguments 1:**

- `patternName` - The name of the assignment pattern (for example, ROUND_ROBIN).
- `groupName` - The name of the group from which to select a subordinate user.
- `realmName` - The name of the identity service realm to which the group belongs.
- `patternParam1 ... patternParamN` - Any additional parameters required by the assignment pattern implementation (may be optional, depending on the pattern).

**Signature 2:**

```java
hwf:wfDynamicUserAssign(patternName,userList,realmName?,patternParam1?,patternParam2?,...,patternParamN?)
```

**Arguments 2:**

- `patternName` - The name of the assignment pattern (for example, ROUND_ROBIN).
- `userList` - The list of users from which to select a user.
- `realmName` - The name of the identity service realm to which the users belong.
- `patternParam1 ... patternParamN` - Any additional parameters required by the assignment pattern implementation (may be optional, depending on the pattern).

**Property IDs:**

- `namespace-uri: http://xmlns.oracle.com/bpel/workflow/xpath`
- `namespace-prefix: hwf`

**See Also:** "Dynamic Assignment Functions" on page 15-113

---

**String Functions**

This section describes the following functions.

- `compare`
- `compare-ignore-case`
- `create-delimited-string`
String Functions

- ends-with
- format-string
- get-content-as-string
- get-localized-string
- index-within-string
- last-index-within-string
- left-trim
- lower-case
- matches
- right-trim
- upper-case

**compare**

This function returns the lexicographical difference between `inputString` and `compareString` comparing the unicode value of each character of both the strings.

This function returns -1 if `inputString` lexicographically precedes the `compareString`.

This function returns 0 if both `inputString` and `compareString` are equal.

This function returns 1 if `inputString` lexicographically follows the `compareString`.

**Example:** `xp20:compare('Audi', 'BMW')` returns -1

**Signature:**

`xp20:compare(inputString as string, compareString as string)`

**Arguments:**

- `variableName` - The source variable for the data
- `propertyName` - The qualified name (QName) of the property

**Property IDs:**

- `namespace-uri`: http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.XPath20
- `namespace-prefix`: `xp20`

**compare-ignore-case**

This function returns the lexicographical difference between `inputString` and `compareString` while ignoring case and comparing the unicode value of each character of both the strings.

This function returns -1 if `inputString` lexicographically precedes the `compareString`.

This function returns 0 if both `inputString` and `compareString` are equal.

This function returns 1 if `inputString` lexicographically follows the `compareString`.

D-34  Oracle BPEL Process Manager Developer’s Guide
Example: `xp20:compare-ignore-case('Audi','bmw')` returns -1

Signature:

```xml
xp:compare-ignore-case(inputString as string, compareString as string)
```

Arguments:
- `inputString` as `string` - The input string
- `CompareString` as `string` - The string to compare against the input string

Property IDs:
- `namespace-uri`:
- `namespace-prefix`: `xp20`

---

**create-delimited-string**

This function returns a delimited string created from `nodeSet` delimited by `delimiter`.

Signature:

```xml
orcl:create-delimited-string(nodeSet as node-set, delimiter as string)
```

Arguments:
- `nodeSet` - The node set to be converted into a delimited string
- `delimiter` - The character that separates the items in the output string; for example, a comma or a semicolon.

Property IDs:
- `namespace-uri`:
- `namespace-prefix`: `orcl`

---

**ends-with**

This function returns true if `inputString` ends with `searchString`.

Example: `xp20:ends-with('XSL Map','Map')` returns true

Signature:

```xml
xp20:ends-with(inputString as string, searchString as string)
```

Arguments:
- `inputString` - The string of data to be searched
- `searchString` - The string for which the function searches

Property IDs:
- `namespace-uri`:
- `namespace-prefix`: `xp20`
**format-string**

This function returns the message formatted with the arguments passed. At least one argument is required and supports up to a maximum of 10 arguments.

**Example:** `orcl:format-string('{0} + {1} = {2}', '2', '2', '4')` returns '2 + 2 = 4'

**Signature:**

`orcl:format-string(string, string, string...)`

**Arguments:**

- `string` - One of the strings to be used in the formatted output

**Property IDs:**

- `namespace-prefix`: orcl

**get-content-as-string**

This function returns the XML representation of the input element.

**Signature:**

`orcl:get-content-as-string(element as node-set)`

**Arguments:**

- `element` - The input element that the function returns as an XML representation

**Property IDs:**

- `namespace-prefix`: orcl

**get-localized-string**

This function returns the locale-specific string for key. This function uses language, country, variant, and resource bundle to identify the correct resource bundle.

The resource bundle in obtained by resolving `resourceLocation` against the `resourceBaseURL`. The URL is assumed to be a directory only if it ends with `/`.

**Usage:**

`orcl:get-localized-string(resourceBaseURL as string, resourceLocation as string, resource bundle as string, language as string, country as string, variant as string, messageKey as string)`

**Example:**

`orcl:get-localized-string('file:/c/','MyResourceBundle','en', 'US', '', 'MSG_KEY')` returns a locale-specific string from a resource bundle 'MyResourceBundle' in the C:\ directory

**Signature:**

`orcl:get-localized-string(resourceURL, resourceLocation, resourceName, language, country, variant, messageKey)`
String Functions

Arguments:
- resourceURL - The URL of the resource
- resourceLocation - The subdirectory location of the resource
- resourceBundleName - The name of the zip file containing the resource bundle
- language - The language of the localized output
- country - The country of the localized output
- variant - The language variant of the localized output
- messageKey - The message key in the resource bundle

Property IDs:
- namespace-prefix: orcl

See Also: "Configuring Messages in Different Languages" on page 15-83 for an example of using this function

index-within-string
This function returns the zero-based index of the first occurrence of searchString within the inputString.
This function returns -1 if searchString is not found.
Example: orcl:index-within-string('ABCABC', 'B') returns 1

Signature:
orcl:index-within-string(inputString as string, searchString as string)

Arguments:
- inputString - The string to be searched
- searchString - The string for which the function searches in the inputString

Property IDs:

last-index-within-string
This function returns the zero-based index of the last occurrence of searchString within inputString.
This function returns -1 if searchString is not found.
Example: orcl:last-index-within-string('ABCABC', 'B') returns 4

Signature:
String Functions

**orcl:last-index-within-string**

*Function Definition:*

```
orcl:last-index-within-string(inputString as string, searchString as string)
```

**Arguments:**

- **inputString**: The string to be searched
- **searchString**: The string for which the function searches in the `inputString`

**Property IDs:**

- **namespace-uri**: http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.ExtFunc
- **namespace-prefix**: `orcl`

**left-trim**

This function returns the value of `inputString` after removing all the leading white spaces.

**Example:**

```
orcl:left-trim('  account  ') returns 'account '
```

**Signature:**

```
orcl:left-trim(inputString)
```

**Arguments:**

- **inputString**: The string to be left-trimmed

**Property IDs:**

- **namespace-uri**: http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.ExtFunc
- **namespace-prefix**: `orcl`

**lower-case**

This function returns the value of `inputString` after translating every character to its lower-case correspondent.

**Example:**

```
xp20:lower-case('ABc!D') returns 'abc!d'
```

**Signature:**

```
xp20:lower-case(inputString)
```

**Arguments:**

- **inputString**: The input string

**Property IDs:**

- **namespace-uri**: http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.XPath20
- **namespace-prefix**: `xp20`

**See Also:** "Chaining Functions" on page 13-10 for an example of using this function in the XSLT Mapper
matches

This function returns true if inputString matches the regular expression pattern regexPattern.

Example: xp20:matches('abracadabra', '^a.*a$') returns true

Signature:
xp20:matches(inputString, regexPattern)

Arguments:
- inputString - The input string
- regexPattern - The regular expression pattern

Property IDs:
- namespace-uri:
  http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.Xpath20
- namespace-prefix: xp20

right-trim

This function returns the value inputString after removing all the trailing white spaces.

Example: orcl:right-trim(' account ') returns ' account'

Signature:
orcl:right-trim(inputString as string)

Arguments:
- inputString - The input string to be right-trimmed

Property IDs:
- namespace-uri:
  http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.ExtFunc
- namespace-prefix: orcl

See Also: "Chaining Functions" on page 13-10 for an example of using this function in the XSLT Mapper

upper-case

This function returns the value of inputString after translating every character to its upper-case correspondent.

Example: xp20:upper-case('abCd0') returns 'ABCDO'

Signature:
xp20:upper-case(inputString as string)

Arguments:
- inputString - The input string

Property IDs:
Utility Functions

This section describes the following functions.

authenticate

This function authenticates a lightweight directory access protocol (LDAP) user and returns true or false.

Signature:
ldap:authenticate('properties', 'userId', 'password')

Arguments:
- properties - The name of the directory specified in the directories.xml file
- userId - The LDAP user's ID
- password - The LDAP user's password

Property IDs:
- namespace-uri: http://schemas.oracle.com/xpath/extension/ldap
- namespace-prefix: ldap

batchProcessActive

This function returns the number of active processes in the batch.

Signature:
ora:batchProcessActive(String batchId, String processId)

Arguments:
- batchId - The ID of the batch
- processId - The ID of the process

batchProcessCompleted

This function returns the number of completed processes in the batch.

Signature:
ora:batchProcessCompleted(String batchId, String processId)

Arguments:
- batchId - The ID of the batch
- processId - The ID of the process

namespace-prefix: xp20
Utility Functions

XPath Extension Functions

Property IDs:
- namespace-uri: http://schemas.oracle.com/xpath/extension
- namespace-prefix: ora

format
This function formats a message using Java’s message format.
Signature:
ora:format(formatStrings, args+)
Arguments:
- formatStrings - The string of data to be formatted
- args+ -

Property IDs:
- namespace-uri: http://schemas.oracle.com/xpath/extension
- namespace-prefix: ora

genEmptyElem
This function generates a list of empty elements for the given QName.
Signature:
ora:genEmptyElem('ElemQName',size?, 'TypeQName'? , xsiNil?)
Arguments:
- ElemQName - The first argument is the QName of the empty elements
- size - The second optional integer argument for the number of empty elements. If missing, the default size is 1.
- TypeQName - The third optional argument is the QName, which is the xsi:type of the generated empty name. This xsi:type pattern matches SOAPENC:Array. If missing or an empty string, the xsi:type attribute is not generated.
- xsiNil - The fourth optional Boolean argument is to specify whether the generated empty elements are XSI - nil, provided the element is XSD-nillable. The default is false. If missing or false, xsi:nil is not generated.

Property IDs:
- namespace-uri: http://schemas.oracle.com/xpath/extension
- namespace-prefix: ora

See Also: "Generating Functionality Equivalent to an Array of an Empty Element" on page 3-20

getFirstChild
This function gets a child element for the given element.
Signature:
ora:getFirstChild(element, index)
Arguments:

XPath Extension Functions  D-41
Utility Functions

- element - The source for the data
- index - The integer value of the child element index

Property IDs:

- namespace-uri: http://schemas.oracle.com/xpath/extension
- namespace-prefix: ora

getMessage

This function gets a message based on the arguments.

Signature:

ora:getMessage(locale, relativeLocation, resourceName, resourceKey, resourceLocation?)

Arguments:

- locale - The locale of the message
- relativeLocation - The subdirectory or message
- resourceName - The name of the message resource
- resourceKey - The key of the resource
- resourceLocation - The location of the resource

Property IDs:

- namespace-uri: http://schemas.oracle.com/xpath/extension
- namespace-prefix: ora

listUsers

This function returns a list of LDAP users.

Signature:

ldap:listUsers('properties', 'filter')

Arguments:

- properties - The properties name as defined in the directories.xml file
- filter - The LDAP filter

Property IDs:

- namespace-uri: http://schemas.oracle.com/xpath/extension/ldap
- namespace-prefix: ldap

max-value-among-nodeset

This function returns the maximum value from a list of input numbers, the node-set inputNumber.

The node-set inputNumber can be a collection of text nodes or elements containing text nodes.
In the case of elements, the first text node's value is considered.

Signature:
Utility Functions

XPath Extension Functions

**orcl:max-value-among-nodeset**

```xml
orcl:max-value-among-nodeset(inputNumber as node-set)
```

**Arguments:**
- **inputNumber** - The node-set of input numbers

**Property IDs:**
- namespace-prefix: orcl

**min-value-among-nodeset**

This function returns the minimum value from a list of input numbers, the node-set `inputNumbers`.
The node-set can be a collection of text nodes or elements containing text nodes.
In the case of elements, the first text node's value is considered.

```xml
orcl:min-value-among-nodeset(inputNumbers as node-set)
```

**Arguments:**
- **inputNumber** - The node-set of input numbers

**Property IDs:**
- namespace-prefix: orcl

**search**

This function returns a list of LDAP entries.

```xml
ldap:search('properties','filter','scope'?)
```

**Arguments:**
- **properties** - The properties name as defined in the bpel.xml file
- **filter** - The filter for the entries
- **scope** - The scope of the search

**Property IDs:**
- namespace-uri: http://schemas.oracle.com/xpath/extension/ldap
- namespace-prefix: ldap

**square-root**

This function returns the square root of `inputNumber`.

```xml
orcl:square-root(25)
```

**Example:** orcl:square-root(25) returns 5

**Signature:**
Utility Functions

orcl:square-root(inputNumber as number)

Arguments:
■ inputNumber - The input number for which the function calculates the square root

Property IDs:
■ namespace-prefix: orcl

translateFromNative
This function translates the input stream to an XML file.
Signature:
ora:translateFromNative('string','nxsdTemplate'?,'nxsdRoot'?)

Arguments:
■ string - The data to be converted into an XML file.
■ nxsdTemplate - The XSD file used to define how the translation is performed.
■ nxsdRoot - The root element defined in the XSD file.

Property IDs:
■ namespace-uri: http://schemas.oracle.com/xpath/extension
■ namespace-prefix: ora

translateToNative
Translates the XML to the native data.
Signature:
ora:translateFromNative('string','nxsdTemplate'?,'nxsdRoot'?)

Arguments:
■ string - The XML file to be converted into a string.
■ nxsdTemplate - The XSD file used to define how the translation is performed.
■ nxsdRoot - The root element defined in the XSD file.

Property IDs:
■ namespace-uri: http://schemas.oracle.com/xpath/extension
■ namespace-prefix: ora
Adding a Custom XPath Function

This section describes how you can create and add your own XPath functions to Oracle BPEL Process Manager. In this example, a function named getNodeValue(arg1) is created.

1. Implement the com.oracle.bpel.xml.xpath.IXPathFunction interface for your XPath function. The IXPathFunction interface has one method named call(context, args). The signature of this method is as follows:

   Object call(IXPathContext context, List args)

   throws XPathFunctionException;

   For the following example, a function named getNodeValue(arg1) is implemented that gets a value of w3c node:

   package com.collaxa.cube.xml.xpath.dom.functions;
   public class GetNodeValue implements IXPathFunction {
       Object call(IXPathContext context, List args)
           throws XPathFunctionException {
           org.w3c.dom.Node node = (org.w3c.dom.Node) args.get(0);
           return node.getNodeValue();
       }
   }

   You now register the getNodeValue XPath function with Oracle BPEL Server.

2. Open the following file:

   SOA_Oracle_Home\bpel\system\config\xpath-functions.xml

3. Add the getNodeValue XPath function under the <bpel-xpath-functions> section.

   <function id="getNodeValue" arity="1">
       <classname>com.collaxa.cube.xml.xpath.dom.functions.GetNodeValueFunction</classname>
       <property id="namespace-uri">
           <value>http://boo.com/xpath/function</value>
           <comment>Namespace URI for this function</comment>
       </property>
       <property id="namespace-prefix">
           <value>boo</value>
           <comment>Namespace prefix for this function</comment>
       </property>
   </function>

4. Note the following function syntax details:

   ■ <function> has the following attributes:
   
   - id — Defines the XPath function name
   - arity — Defines the number of required arguments that the function can accept (for example, a function takes 5 arguments, but the last 2 arguments are optional; in this case, the value for arity is 3).

   ■ <classname> — Defines the XPath implementation class name.

5. Specify the following initialization properties for each function:

   ■ namespace-uri — Associate each XPath with a namespace.
   ■ namespace-prefix — Specify a namespace prefix for this XPath function.
Adding a Custom XPath Function

For example, assume you want to call the XPath function from Oracle BPEL Process Manager. You must define the namespace uri in the <process> section as follows:

```xml
<process name="XPathFunction"
  targetNamespace="http://samples.cxdn.com"
  suppressJoinFailure="yes"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:boo="http://boo.com/xpath/function"
  xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process">
  <copy>
    <from expression="boo:getNodeValue(bpws:getVariableData('output','payload','/invoice/lineItems/item[3]/@sku'))"/>
    <to variable="mySKU"/>
  </copy>
</process>
```

6. Go to the following directory to see how this function is used in a sample:

   $OA_Oracle_Home/bpel/samples/references/XPathFunction

7. Review the `getNodeValue()` XPath function code shipped with this sample:

   ```java
   package com.collaxa.cube.xml.xpath.dom.functions;
   import java.util.List;
   import com.oracle.bpel.xml.xpath.IXPathFunction;
   import com.oracle.bpel.xml.xpath.IXPathContext;
   import com.oracle.bpel.xml.xpath.XPathFunctionException;
   import org.w3c.dom.Node;
   
   /**
    * Returns the value of the dom node as String
    * Usage: String ora:getNodeValue(Node node)
    * Node - dom node
    */
   public class GetNodeValueFunction implements IXPathFunction {
      public Object call(IXPathContext context, List args) throws XPathFunctionException {
         try {
            if (args.size() == 1) {
               return evaluate( args.get(0 ) );
            }
         } catch(Exception e) {
            throw new XPathFunctionException (e.getMessage());
         }
         throw new XPathFunctionException ("getNodeValue() requires one argument which is a dom node.");
      }
   }
   ```
public Object evaluate(Object node) throws Exception {
    if (node != null) {
        if (node instanceof Node)
            return ((Node) node).getNodeValue();
        else
            return String.valueOf(node);
    }
    return null;
}
Summary

This appendix lists the XPath extension functions, along with their descriptions, signature, argument descriptions, and property ID information.
Workflow Services Changes Between 10.1.2 and 10.1.3.1

This appendix provides an overview of changes to workflow services between releases 10.1.2 and 10.1.3.1.

This appendix contains the following topics:

- Backwards Compatibility between 10.1.2 and 10.1.3.1
- Changes Between the Workflow Wizard and the Human Task Editor
- Changes to Configuration Files
- Changes to Worklist APIs
- Changes to Task Display Form Deployment
- Changes to the Oracle BPEL Worklist Application
- Changes to Oracle BPEL Control
- Migrating Workflow Definitions from 10.1.2 to 10.1.3.1

See Also:

- "Oracle BPEL Process Manager Workflow Services 10.1.2 and 10.1.3.1.0 Compatibility" on page 15-2 for additional details
- "Deprecated Workflow Service and Identity Service Functions" on page 15-123
- Oracle BPEL Process Manager Installation Guide for additional details about migrating projects that use the workflow service

Backwards Compatibility between 10.1.2 and 10.1.3.1

Workflows designed with the workflow wizard in 10.1.2 can be deployed and run in 10.1.3.1. You just have to redeploy the process. However, the instance tables for 10.1.2 and 10.1.3.1 are different. Therefore, 10.1.2 items do not show up in the 10.1.3.1 Oracle BPEL Worklist Application. To access tasks created from 10.1.2 processes, visit the old Oracle BPEL Worklist Application at http://localhost:9700/integration/oldworklistapp/Login

The identity service configuration file must be modified as described "Migrating Workflow Definitions from 10.1.2 to 10.1.3.1" on page E-11. This modification is required even if you do not make any changes to your 10.1.2 processes. If you want to use any new 10.1.3.1 workflow features, you must manually migrate the process to use...
the 10.1.3 task definition. The changes to the task editor, form deployment, worklist, and configuration files are documented in later sections of this appendix.

## Changes Between the Workflow Wizard and the Human Task Editor

For release 10.1.3.1, the Workflow Wizard that you used to design workflows in previous releases has been replaced with the Human Task Editor. You can still deploy 10.1.2 processes that include workflow functionality from Oracle JDeveloper. However, you cannot edit these workflows with the Human Task Editor. To use the 10.1.3.1 functionality, you must manually migrate the workflow task scope and recreate the human task.

Table 22–1 describes the functionality of the 10.1.2 Workflow Wizard pages and the location of this functionality in the Human Task Editor.

<table>
<thead>
<tr>
<th>10.1.2 Workflow Wizard Page</th>
<th>Location in 10.1.3.1 Human Task Editor</th>
<th>See Section</th>
</tr>
</thead>
</table>
| Welcome — For creating a workflow model, performing advanced task configuration, and extending the functionality of an existing workflow | To create a human task:  
1. Access the Human Task Editor.  
2. Go to the Advanced tab of the human task activity.  
3. Click the Include task history from check box. | "Accessing the Human Task Editor" on page 15-13 to create a human task |
| Note: The pages that display if you select advanced task configuration are described later in this table. |                                                                                  | "Including the Task History of Other Human Tasks" on page 15-59 to extend an existing workflow |
| Workflow Pattern — For selecting a workflow pattern to design, specifying a workflow name, and specifying a variable name | To select a workflow pattern to design.  
1. Go to the Assignment and Routing Policy section.  
2. Click the + sign to display the Add Participant Type window.  
3. Select a participant type from the Type list.  
To specify a workflow name:  
1. Right-click the BPEL process and select Create Human Task Definition.  
2. Specify a name in the Human Task Name field of the Add a Human Task window.  
   or  
1. Go to the General tab of the human task activity.  
2. Specify a name in the Task Definition field.  
To specify a variable name:  
1. Go to the General tab of the human task activity.  
2. Click the flashlight in the BPEL Variable column to map the task parameter to the BPEL variable. | "Assigning Task Participants" on page 15-22 to select a workflow pattern to design  
"Accessing the Human Task Editor" on page 15-13 or "Associating a Human Worklist Task with a BPEL Process" on page 15-54 to specify a workflow name  
"Specifying Task Parameters" on page 15-56 to specify a variable name |
| Task Notifications — For selecting the status for sending a notification message to a recipient |  
1. Go to the Notification Settings section.  
2. Click the + sign to expand the section.  
3. Specify the task notification status. | "Specifying Participant Notification Preferences" on page 15-43 |
Changes Between the Workflow Wizard and the Human Task Editor

Table 22–1 (Cont.) Location of Workflow Wizard Functionality in 10.1.3.1

<table>
<thead>
<tr>
<th>10.1.2 Workflow Wizard Page</th>
<th>Location in 10.1.3.1 Human Task Editor</th>
<th>See Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task Details — For selecting task details such as task title, payload, payload display format, task creator, and expiration duration</td>
<td>To specify a task title: 1. Go to the top of the Human Task Editor. 2. Specify a name in the Title field. or 1. Go to the General tab of the human task activity. 2. Specify a name in the Task Title field. To specify a payload: 1. Go to the Parameters section. 2. Click the + sign to display the Add Task Parameter window. To specify a payload display format: 1. Complete BPEL process design. 2. Right-click the human task folder. 3. Select a payload display format option. To specify a task creator: 1. Go to the General tab of the human task activity. 2. Specify a task creator in the Initiator field. To specify the expiration duration: 1. Go to the Assignment and Routing Policy section. 2. Click the + sign to display the Add Participant Type window. 3. Select a participant type from the Type list. 4. Click Advanced at the bottom for the selected participant type. 5. Provide specific details.</td>
<td>&quot;Specifying a Task Title and Priority&quot; on page 15-16 or &quot;Specifying the Task Title&quot; on page 15-55 to specify a task title. &quot;Specifying the Task Payload Data Structure&quot; on page 15-21 to specify a payload. &quot;Selecting a Task Display Form&quot; on page 15-66 to specify a payload display format. &quot;Specifying the Task Initiator and Task Priority&quot; on page 15-55 to specify a task creator. &quot;Specifying a Time Limit for Acting on a Task&quot; on page 15-26 to specify the expiration duration for a single approver. &quot;Specifying a Time Limit for Acting on a Task&quot; on page 15-29 to specify the expiration duration for a group vote. &quot;Specifying a Time Limit for Acting on a Task&quot; on page 15-32 to specify the expiration duration for a management chain. &quot;Specifying a Time Limit for Acting on a Task&quot; on page 15-34 to specify the expiration duration for a sequential list of approvers.</td>
</tr>
</tbody>
</table>

Task Outcomes — For selecting a possible outcome for the task | 1. Click the icon to the right of the Outcomes field at the top of the Human Task Editor. | "Specifying a Task Outcome" on page 15-16. |
### Changes Between the Workflow Wizard and the Human Task Editor

<table>
<thead>
<tr>
<th>10.1.2 Workflow Wizard Page</th>
<th>Location in 10.1.3.1 Human Task Editor</th>
<th>See Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assignee — For manually or dynamically selecting the user or group to whom to assign a task</td>
<td>Go to the Assignment and Routing Policy section. Click the + sign to display the Add Participant Type window. Select a participant type from the Type list. Select a method for manually or dynamically assigning the user or group to a task.</td>
<td>“Assigning Participants to the Single Approver Task” on page 15-25 “Assigning Participants to the Group Vote Task” on page 15-28 “Assigning Participants to the Management Chain Task” on page 15-33 “Assigning Participants to the Sequential List of Approvers Task” on page 15-33 “Configuring the FYI Assignee Participant Type” on page 15-35 “Configuring the External Routing Service Participant Type” on page 15-36</td>
</tr>
<tr>
<td>Escalation policy — For selecting the method for escalating a task</td>
<td>Go to the Expiration and Escalation Policy section. Click the + sign to expand the section. Select Escalate after from the list. Provide specific details.</td>
<td>“Overview or Escalation and Expiration Policy” on page 15-39 and “Escalate After Policy” on page 15-42</td>
</tr>
<tr>
<td>Renewal policy — For selecting a task renewal policy</td>
<td>Go to the Expiration and Escalation Policy section. Click the + sign to expand the section. Select Renew after from the list. Provide specific details.</td>
<td>“Renew After Policy” on page 15-42</td>
</tr>
<tr>
<td>Assignment policy — For selecting if a specific person or a specific number of people must act on the task. This lets you route tasks to multiple users in a sequence</td>
<td>Go to the Assignment and Routing Policy section. Click the + sign to display the Add Participant Type window. Select Sequential List of Approvers from the list. Specify the list of sequential approvers.</td>
<td>“Assigning Participants to the Sequential List of Approvers Task” on page 15-33</td>
</tr>
<tr>
<td>Routing policy — For selecting the method by which to route the task</td>
<td>Go to the Notifications section. Click the + sign to expand the section. Specify a routing method.</td>
<td>“Notifying Recipients of Changes to Task Status” on page 15-44</td>
</tr>
<tr>
<td>Outcome determination policy — For selecting a task outcome policy, such as the percentage for final outcome, default outcome, and early completion configuration</td>
<td>Go to the Assignment and Routing Policy section. Click the + sign to display the Add Participant Type window. Select the Group Vote participant type from the Type list. Go to the Majority Voted Outcome section. Provide specific details.</td>
<td>“Specifying Group Voting Details” on page 15-29</td>
</tr>
</tbody>
</table>
Management Chain Parameters — For selecting the management chain (initial assignee and other users such as a manager) to sequentially review this task

1. Go to the Assignment and Routing Policy section.
2. Click the + sign to display the Add Participant Type window.
3. Select Management Chain from the list.
4. Specify the number of management chains in the Number of Approvers section.

Reviewers — For manually or dynamically selecting the user or group to review this task (appeared only if the parallel workflow with final reviewer pattern was selected)

Model a parallel participant followed by a simple participant, which acts as the reviewer.

(Advanced Options) Optional Task Details — For selecting the task priority, task owner, and task identification key

To specify the task priority:
1. Go to the top of the Human Task Editor.
2. Select a priority from the Priority list.
   or, to override this setting:
1. Go to the General tab of the human task activity.
2. Select a priority from the Priority list.
To specify the task owner:
1. Go to the top of the Human Task Editor.
2. Specify an owner in the Owner field.
   or, to override this setting:
1. Go to the Advanced tab of the human task activity.
2. Specify an owner in the Owner field.
To specify the identification key:
1. Go to the Advanced tab of the human task activity.
2. Specify the identification key in the Identification Key field.

Table 22–1 (Cont.) Location of Workflow Wizard Functionality in 10.1.3.1

<table>
<thead>
<tr>
<th>10.1.2 Workflow Wizard Page</th>
<th>Location in 10.1.3.1 Human Task Editor</th>
<th>See Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>Management Chain Parameters — For selecting the management chain (initial assignee and other users such as a manager) to sequentially review this task</td>
<td>1. Go to the Assignment and Routing Policy section. 2. Click the + sign to display the Add Participant Type window. 3. Select Management Chain from the list. 4. Specify the number of management chains in the Number of Approvers section.</td>
<td>“Specifying the Number of Approvers” on page 15-31</td>
</tr>
<tr>
<td>Reviewers — For manually or dynamically selecting the user or group to review this task (appeared only if the parallel workflow with final reviewer pattern was selected)</td>
<td>Model a parallel participant followed by a simple participant, which acts as the reviewer.</td>
<td></td>
</tr>
<tr>
<td>(Advanced Options) Optional Task Details — For selecting the task priority, task owner, and task identification key</td>
<td>To specify the task priority: 1. Go to the top of the Human Task Editor. 2. Select a priority from the Priority list. or, to override this setting: 1. Go to the General tab of the human task activity. 2. Select a priority from the Priority list. To specify the task owner: 1. Go to the top of the Human Task Editor. 2. Specify an owner in the Owner field. or, to override this setting: 1. Go to the Advanced tab of the human task activity. 2. Specify an owner in the Owner field. To specify the identification key: 1. Go to the Advanced tab of the human task activity. 2. Specify the identification key in the Identification Key field.</td>
<td>“Specifying the Task Title, Priority, Outcome, and Owner” on page 15-15 or “Specifying the Task Initiator and Task Priority” on page 15-55 to specify the task priority “Specifying a Task Owner” on page 15-58 to specify the task owner “Specifying an Identification Key” on page 15-59 to specify the identification key</td>
</tr>
</tbody>
</table>
Changes to Configuration Files

Table 22–2 Configuration File Changes

<table>
<thead>
<tr>
<th>Configuration</th>
<th>File</th>
<th>Changes Between 10.1.2 and 10.1.3.1</th>
</tr>
</thead>
</table>
| E-mail server   | ns_emails.xml               | This file includes a new property called NotificationMode, which is used for both e-mail and wireless notifications: <EmailAccounts xmlns="http://xmlns.oracle.com/ias/pcbpel/NotificationService" EmailMimeCharset="" NotificationMode="NONE"> Set this property as follows (NONE is the default):  
  - EMAIL — If you only want to set up e-mail, and not other channels such as voice, SMS, and so on  
  - ALL — If you want to set up e-mail and all other channels, such as voice, SMS, and so on  
  The note in this file explains the settings in detail. |
| Wireless        | ns_iaswconfig.xml           | No changes.                          |
| Fax             | ns_faxcoverpages.xml        | No changes.                          |
| Workflow properties |wf_config.xml               | This file now includes workflow property settings that previously appeared in pc.properties. |

E-6 Oracle BPEL Process Manager Developer’s Guide
Changes to Worklist APIs

In 10.1.3.1, a new set of APIs is provided for interacting with workflow services. These APIs only support 10.1.3.1 workflow tasks, and cannot be used to query or update 10.1.2 tasks or task definitions.

For interaction with 10.1.2 tasks, the 10.1.2 APIs are still provided. These APIs can query and update 10.1.2 tasks and task definitions. However, they cannot be used for 10.1.3.1 tasks.

Table 22-3 describes the worklist API changes between 10.1.2 and 10.1.3.1.
Changes to Task Display Form Deployment

Table 22–3  API Worklist Changes

<table>
<thead>
<tr>
<th>Component</th>
<th>10.1.2</th>
<th>10.1.3.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>API Clients</td>
<td>Different APIs to support local Java client and remote EJB client are provided.</td>
<td>Several API clients (Java, local and remote EJB, and SOAP) are provided. Local and remote EJBs are now unified in a single API. To use the same API, modify the <code>wf_client_config.xml</code> file to use the appropriate communication protocol. The <code>wf_client_config.xml</code> file is populated by default at install time with the EJB and SOAP configurations necessary to switch between EJB and SOAP bindings.</td>
</tr>
</tbody>
</table>

See Also: “Building Clients for Workflow Services” on page 16-54

Packages that contain the APIs:
- `oracle.tip.pc.api.worklist` — The workflow client API
- `oracle.bpel.services.workflow.metadata` — Corresponds to the task metadata service, which exposes operations to retrieve metadata information related to a task
- `oracle.bpel.services.workflow.query` — Corresponds to the task query service, which queries tasks based on a variety of search criterion such as keyword, category, status, business process, attribute values, history information of a task, and so on
- `oracle.bpel.services.workflow.report` — Corresponds to the report query service, which creates reports such as unattended tasks, tasks priority, tasks cycle time, and tasks productivity
- `oracle.bpel.services.workflow.runtimeconfig` — Corresponds to the runtime config service, which manages metadata used in the task service run-time environment. It principally supports management of task payload flex field mappings.
- `oracle.bpel.services.workflow.task` — Corresponds to the task service, which exposes operations to act on tasks
- `oracle.bpel.services.workflow.user` — Corresponds to the user metadata service, which provides methods for managing metadata specific to individual users and groups

Changes to Task Display Form Deployment

Table 22–4 describes the task display form deployment changes between 10.1.2 and 10.1.3.1.
### Changes to Task Display Form Deployment

**Workflow Services Changes Between 10.1.2 and 10.1.3.1**

#### Table 22–4 Task Display Form Deployment

<table>
<thead>
<tr>
<th>Issue</th>
<th>10.1.2</th>
<th>10.1.3.1</th>
</tr>
</thead>
</table>
| Specifying the task display form | You specify the task display form to use during workflow design in the Workflow Wizard - Task Details window:  
- Auto generate JSP Form  
- XSL File  
- JSP URL | You specify the task display form to use after completing human task design:  
- Auto Generate Simple Task Form  
- Custom Task Form  
See Also: "Task 3: Generating the Task Display Form" on page 15-66 |
| Contents of the autogenerated JSP task form | Automatically creates two files to display the payload:  
- `task_name_WF_Form.jsp` — a default JSP file  
- `task_name_WF_Fields.xml` — a mapping file | The default layout is based on a three-region template:  
- Header region — Displays task attributes such as title, priority, created date, assignee, and expiration date. This information is contained in the `Header1.jsp` file.  
- Body region — Displays content based on task parameters. The information is contained in the `payload-body.jsp` file and `payload-body.xml` mapping file.  
- Footer region — Displays comments, attachments, and a short history of the task routing. This information is contained in the `Footer1.jsp` file. A `.tform` file is also generated, which includes a template URL and region information. This file is part of the process deployment archive and is deployed during process deployment.  
See Also: "Automatically Generating a Simple Task Display Form" on page 15-68 |
| Contents of the custom JSP task form | You write a JSP for payload presentation display in Oracle BPEL Worklist Application | Task display forms are generated using three templates and two default JSPs:  
- Three Region JSP — Consists of header, body, and footer regions. These regions can be displayed using custom JSP, XSL, default JSP, or autogenerated JSP files. The automatically generated JSP displays the body region.  
- Two Region JSP — Consists of the header and footer regions  
- One Region JSP — Consists of the body region  
The two default JSPs are as follows:  
- The header JSP displays task attributes such as task number, priority, title, and so on.  
- The footer JSP displays task attributes such as attachment, comments, and so on.  
See Also: "Generating a Custom Task Display Form" on page 15-74 |

**Workflow Services Changes Between 10.1.2 and 10.1.3.1**

E-9
Changes to the Oracle BPEL Worklist Application

Table 22–5 lists the URLs for accessing the old and new Oracle BPEL Worklist Application. You access tasks created from 10.1.2 processes by visiting the old Oracle BPEL Worklist Application URL. You access tasks designed in 10.1.3.1 with the Human Task editor by visiting the new URL.

<table>
<thead>
<tr>
<th>Issue</th>
<th>10.1.2 Worklist Application URL</th>
<th>10.1.3.1 Worklist Application URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Worklist code</td>
<td>SOA_Oracle_</td>
<td>SOA_Oracle_</td>
</tr>
<tr>
<td>Location</td>
<td>j2ee\home\applications\hw\services\worklistexpress\pay load\bpel\WorkflowName_Version.</td>
<td>j2ee\home\applications\default\Process_Name_Version\Workflow_Name.</td>
</tr>
</tbody>
</table>

See Also: “Customizing the Worklist Application” on page 16-41

Changes to Oracle BPEL Control

In 10.1.2, the workflow definition was defined in the following locations:

- A configuration file
- As BPEL code inside the workflow scope

The generated BPEL code varied based on the workflow pattern you selected to design the process. The BPEL code in the process also used the task manager process for some of its functionality. Therefore, for every process that included human workflow, the audit trail showed the activities executed inside the scope. Also, every instance of the task included a corresponding instance of the TaskActionHandler BPEL process.
In 10.1.3.1, workflows are completely metadata-driven. Therefore, the generated BPEL code is much simpler; it consists of several assigns followed by invoke and receive activities from the task manager service. The task manager is no longer implemented as a BPEL process. Therefore, the audit trail shows a simplified view of the task. Also, the audit trail has been enhanced to show the task history. Therefore, clicking the task activity queries the current state of the task and displays the status and assignees.

**Migrating Workflow Definitions from 10.1.2 to 10.1.3.1**

1. Identify the scope activities that include human workflows.
2. Create a new human task definition for each of the workflows identified in Step 1.
3. Give it the same name as the old scope.
4. Drag and drop a human task activity in your BPEL process.
   This creates a new 10.1.3.1 scope activity and switch activity.
5. Remove the 10.1.2 scope activity. However, do not remove the 10.1.2 switch statement.
6. Copy the case statements from the 10.1.3.1 switch to the 10.1.2 switch and retain any of the code that you have in the case blocks.
7. Remove the 10.1.3.1 switch activity.
8. Remove the 10.1.2 global task variable. When the human task is created in 10.1.3.1, a new variable is created.
9. Identify task forms used for the workflow by looking for the _form.xml and .jsp files.
   In 10.1.2, task forms are displayed by using the autogenerated JSP form, the XSLT template, and the user-defined JSP URL. For the autogenerated JSP form, you can use the 10.1.3.1 autogenerated simple task form. For the XSLT template and user-defined JSP URL you can use the 10.1.3.1 custom task form.
   With the 10.1.3.1 custom task form, you must select the three region JSP template, which includes the header, payload, and footer regions. For the header and footer regions, you can use the default header and footer JSP, respectively. For the payload, you can use the XSLT or JSP type based on the 10.1.2 task form display.
10. Regenerate the task forms.
11. Redo any customizations that were performed on the old task forms.
Migrating Workflow Definitions from 10.1.2 to 10.1.3.1
Index

Numerics
101.HelloWorld tutorials, 1-10
102.InvokingProcesses tutorials, 1-10
103.XMLDocuments tutorials, 1-10
104.SyncQuoteConsumer tutorials, 1-10
105.AsyncCompositeLoanBroker tutorials, 1-10
106.ParallelFlows tutorials, 1-10
107.Exceptions tutorials, 1-11
108.Timeouts tutorials, 1-11
109.CorrelationSets tutorials, 1-11
112.Arrays tutorials, 1-11
113.ABCARouting tutorials, 1-11
114.XSLTTransformations tutorials, 1-11
115.XQueryTransformations tutorials, 1-11
121.FileAdapter tutorials, 1-11
122.DBAAdapter tutorials, 1-12
123.AQAdapter tutorials, 1-13
124.jmsAdapter tutorials, 1-13
125.ReportsSchema tutorials, 1-14
126.DataAggregator tutorials, 1-14
127.OrderBookingTutorial tutorials, 1-14
128.GoogleFlow tutorials, 1-14
129.FTPAdapter tutorials, 1-14
130.SendEmailWithAttachments tutorials, 1-14
132.UserTasks tutorials, 1-14
133.SecureInvokingProcesses tutorials, 1-14
140.AdapterFramework tutorials, 1-14
150.AppsAdapter tutorials, 1-14
701.LargeProcesses tutorials, 1-14
702.Bindings tutorials, 1-15

A
abs function description, D-23
accountName XML file, 14-15
action permissions
in workflows, 16-17
actionable notification
definition, 14-1
actional e-mails, 15-46, 15-83
activation-agent
customize task syntax, 19-8
e-mail, 14-14
activationAgent attributes
setting for notifications in the bpel.xml file, 14-14
active
process life cycle, 19-17
activities
adapters tab, B-2
annotations tab, B-2
assign, B-3
compensate, B-4
correlation-sets tab, B-2
decide, B-5
definition, 1-3, 2-15
e-mail, B-6
empty, B-8
fax, B-9
flow, B-11
flowN, B-12
human task, B-12
invoke, B-14
Java embedding, B-16
overview, 2-15, B-1
picker, B-17
pick, B-19
receive, B-20
reply, B-21
scope, B-22
sensors tab, B-2
sequence, B-24
SMS, B-25
switch, B-29
tasks common to many activities, B-2
terminate, B-27
throw, B-28
transform, B-29
voice, B-30
wait, B-31
while, B-32
Activities tab
Oracle BPEL Control, 19-27
activity executed ASSERT
overview, 20-3
activity interactions
viewing, 19-26
activity sensors
definition, 17-1
Adapter Configuration wizard
starting, 2-20
adapters
configuring, 2-20
definition, 2-20
documentation, 2-21
in Oracle JDeveloper, 2-20
overview, B-33
service names, 2-20
supported technology adapters, 1-2
supported third-party adapters, 1-2
tutorials for third-party adapters, 1-3
adapters tab
in activities, B-2
add-dayTimeDuration-to-dateTime function
description, D-18
addQuotes function
description, D-5
ADF data controls
preview release support, 15-67
ADT queue
tutorial, 1-13
ADT types supported with 10.1.3 types
tutorial, 1-13
alarm table
refreshing from Oracle BPEL Control, 19-15
AmazonFlow
demos, 1-7
annotations tab
in activities, B-2
and
customize task for deploying to multiple environments with different configuration values, 19-5
definition, 19-28
deploying test suites, 20-20
deploying to multiple environments with different configuration values, 19-5
deploying workflow task display forms, 15-77
deployProcess test, 20-26
generating custom reports, 20-29
generating report results, 20-24
performing local deployment, 19-11
running test suites, 20-24
specifying the property value for development and production environments in a single build file location, 19-5
syntax for customize task, 19-7
task examples, 19-28
using the developer prompt, 2-3
Apache Axis
demos of interoperability with Oracle BPEL Process Manager, 1-7
tutorials, 1-7
annotations tab
in activities, B-2
ant
customize task for deploying to multiple environments with different configuration values, 19-5
definition, 19-28
deploying test suites, 20-20
deploying to multiple environments with different configuration values, 19-5
deploying workflow task display forms, 15-77
deployProcess test, 20-26
generating custom reports, 20-29
generating report results, 20-24
performing local deployment, 19-11
running test suites, 20-24
specifying the property value for development and production environments in a single build file location, 19-5
syntax for customize task, 19-7
task examples, 19-28
using the developer prompt, 2-3
application configuration wizard
starting, 2-20
adapters
configuring, 2-20
definition, 2-20
documentation, 2-21
in Oracle JDeveloper, 2-20
overview, B-33
service names, 2-20
supported technology adapters, 1-2
supported third-party adapters, 1-2
tutorials for third-party adapters, 1-3
application
blank spaces not permitted in directory path, 2-3
creating in Oracle JDeveloper, 2-3
definition, 2-3
AQ adapter
ADT queue tutorial, 1-13
ADT with XMLType payload tutorial, 1-13
capabilities, B-34
CLOB payload as opaque tutorial, 1-13
CLOB payload tutorial, 1-13
correlations tutorial, 1-13
erqueuing files to BLOB columns tutorial, 1-13
listening for message recipients tutorial, 1-13
multiconsumer queues tutorial, 1-13
raw queues tutorial, 1-13
rejectedMessageHandler property tutorial, 1-13
rules-based subscriber tutorial, 1-13
simple XMLType payload tutorial, 1-13
SQL primitive types tutorial, 1-13
supported 10.1.3 ADT types tutorial, 1-13
tutorials, 1-13
AQ adapter BLOB column enqueuing
tutorial, 1-13
AQ adapter correlations
tutorial, 1-13
Application Navigator
accessing the Human Task editor from, 15-13
contents of, 2-6
definition, 2-6
location of in Oracle JDeveloper, 2-5
applications
blank spaces not permitted in directory path, 2-3
creating in Oracle JDeveloper, 2-3
definition, 2-3
Application configuration wizard
starting, 2-20
adapters
configuring, 2-20
definition, 2-20
documentation, 2-21
in Oracle JDeveloper, 2-20
overview, B-33
service names, 2-20
supported technology adapters, 1-2
supported third-party adapters, 1-2
tutorials for third-party adapters, 1-3
adapters tab
in activities, B-2
add-dayTimeDuration-to-dateTime function
description, D-18
addQuotes function
description, D-5
ADF data controls
preview release support, 15-67
ADT queue
tutorial, 1-13
ADT types supported with 10.1.3 types
tutorial, 1-13
alarm table
refreshing from Oracle BPEL Control, 19-15
AmazonFlow
demos, 1-7
Annotations tab
in activities, B-2
and
customize task for deploying to multiple environments with different configuration values, 19-5
definition, 19-28
deploying test suites, 20-20
deploying to multiple environments with different configuration values, 19-5
deploying workflow task display forms, 15-77
deployProcess test, 20-26
generating custom reports, 20-29
generating report results, 20-24
performing local deployment, 19-11
running test suites, 20-24
specifying the property value for development and production environments in a single build file location, 19-5
syntax for customize task, 19-7
task examples, 19-28
using the developer prompt, 2-3
Apache Axis
demos of interoperability with Oracle BPEL Process Manager, 1-7
tutorials, 1-7
Application Navigator
accessing the Human Task editor from, 15-13
contents of, 2-6
definition, 2-6
location of in Oracle JDeveloper, 2-5
applications
blank spaces not permitted in directory path, 2-3
creating in Oracle JDeveloper, 2-3
definition, 2-3
AQ adapter
ADT queue tutorial, 1-13
ADT with XMLType payload tutorial, 1-13
capabilities, B-34
CLOB payload as opaque tutorial, 1-13
CLOB payload tutorial, 1-13
correlations tutorial, 1-13
erqueuing files to BLOB columns tutorial, 1-13
listening for message recipients tutorial, 1-13
multiconsumer queues tutorial, 1-13
raw queues tutorial, 1-13
rejectedMessageHandler property tutorial, 1-13
rules-based subscriber tutorial, 1-13
simple XMLType payload tutorial, 1-13
SQL primitive types tutorial, 1-13
supported 10.1.3 ADT types tutorial, 1-13
tutorials, 1-13
AQ adapter BLOB column enqueuing
tutorial, 1-13
AQ adapter correlations
tutorial, 1-13
Application Navigator
accessing the Human Task editor from, 15-13
contents of, 2-6
definition, 2-6
location of in Oracle JDeveloper, 2-5
applications
blank spaces not permitted in directory path, 2-3
creating in Oracle JDeveloper, 2-3
definition, 2-3
AQ adapter
ADT queue tutorial, 1-13
ADT with XMLType payload tutorial, 1-13
capabilities, B-34
CLOB payload as opaque tutorial, 1-13
CLOB payload tutorial, 1-13
correlations tutorial, 1-13
erqueuing files to BLOB columns tutorial, 1-13
listening for message recipients tutorial, 1-13
multiconsumer queues tutorial, 1-13
raw queues tutorial, 1-13
rejectedMessageHandler property tutorial, 1-13
rules-based subscriber tutorial, 1-13
simple XMLType payload tutorial, 1-13
SQL primitive types tutorial, 1-13
supported 10.1.3 ADT types tutorial, 1-13
tutorials, 1-13
AQ adapter BLOB column enqueuing
tutorial, 1-13
AQ adapter correlations
tutorial, 1-13
AQM adapter message recipients listener tutorial, 1-13
AQM adapter multiconsumer queues tutorial, 1-13
AQM adapter raw queues tutorial, 1-13
AQM adapter SQL primitive types tutorial, 1-13
archive definition, 19-1
arrays determining the size of, 3-17
in transformations, 13-14
manipulating, 3-16
maxOccurs attribute, 3-16
SOAP-encoded arrays not supported, 3-21
statically indexing into, 3-16
tutorials, 1-11
assert facts and execute rule set operation definition, 18-15
assert facts only operation definition, 18-15
assert facts, execute rule set, and retrieve results operation
definition, 18-15
assertions creating activity execution asserts, 20-15
creating value asserts, 20-14
creating XML asserts, 20-15
fatal assertions not supported at runtime, 20-15,
20-16
in BPEL test suites, 20-5
assertXPath customizing in business rules, 18-50
assign activity adding to an asynchronous service, 5-14
capabilities, B-3
copying data, 3-5
description, 3-2
for data manipulation, 3-2
formatting the e-mail message body as HTML, 14-8
in asynchronous services, 5-14
references, 1-9
using multiple bpelex:append settings, B-4
assignment service configuration, 15-113
deploying a custom assignment service, 15-121
dynamic assignment functions, 15-113, 15-114,
15-117, 15-118

dynamically assigning task participants, 15-118
example of implementation, 15-120
implementing, 15-119
overview, 15-119
asynchronous callbacks, 5-3
asynchronous event emulation in BPEL tests, 20-29
asynchronous interaction with a notification timer
BPEL process as the client, 12-6
BPEL process as the service, 12-6
definition, 12-5
asynchronous interaction with timeout
BPEL process as the client, 12-5
BPEL process as the service, 12-5
definition, 12-4
asynchronous interactions
BPEL process as the client, 12-4
BPEL process as the service, 12-4
definition, 12-3
returning faults, 8-7
asynchronous processes
dehydration store, 5-10
retrieving status and results from, 11-5
tutorial, 1-10
using dehydration, 5-10
asynchronous services
assign activities, 5-14
calling, 5-4, 5-11
correlating messages, 5-7
correlation IDs, 5-6
correlation sets, 5-14
invoke activities, 5-5, 5-12
managing multiple instances, 5-7
parallel flows, 6-1
partner links, 5-4, 5-5, 5-11
partnerLinkTypes, 5-3
receive activities, 5-5, 5-13
reply activities, 5-10
sample, 1-8
tutorials, 1-14
use case, 5-1
WS-Addressing, 5-6
attachments binary attachments in SOAP messages, 3-28
sending with the notification wizard, 14-5
task attachments with e-mail notifications, 15-46,
15-84
tutorials, 1-14
using style sheets, 15-48
using WordML style sheets, 15-48
attribute labels internationalization, 15-113
attributes manipulating, 3-9
audit trail viewing, 19-26
authenticate function description, D-40
auto mapping in transformations, 13-15
with confirmation in transformations, 13-17
AutoLoanDemo demos, 1-7
Axis tutorials, 1-18
business rules dictionary
definition, 18-3

C

cache
clearing the WSDL cache, 19-14
callback classes
specifying on task status, 15-51
callbacks
task routing and customization in BPEL
callbacks, 15-52, 15-59
viewing, 15-62
catch branch
creating, B-23
fault handling, 8-8
references, 1-9
tutorial, 1-11
channels
e-mail, 14-4
fax, 14-8
pager, 14-10
SMS, 14-11
voice mail, 14-12
CheckoutDemo
demos, 1-7
class names
specifying in the external routing service
participant type, 15-37
clearTaskAssignees function
description, D-28
client APIs
in BPEL tests, 20-32
CLOB payload
tutorial, 1-13
CLOB payload as opaque
tutorial, 1-13
COBOL Copybook
tutorial, 1-11
code coverage tests
overview, 20-3
comma-separated value (CSV) files
exporting reports to CSV files, 22-3
tutorials, 1-11
compare function
description, D-34
compare-ignore-case function
description, D-34
compatibility
between workflow services 10.1.2 and
10.1.3.1.0, 15-2, E-6
compensate activity
capabilities, B-4
definition, 8-9
fault handling, 8-9
tutorial, 1-7
compilation
of BPEL processes, 19-1
without deployment on Oracle JDeveloper, 19-4
compiling
VacationRequest process, 15-95
complex structures
tutorial, 1-11
complex type
variables, 3-6
Component Palette
accessing the Human Task editor from,
15-14
definition, 2-10
location of in Oracle JDeveloper, 2-5
transformation functions, 2-10
composite test
definition, 20-2
concat function
description, 3-7
conditional branching logic
definition, 7-1
tutorials, 1-14
use case, 7-1
use of XPath expressions, 7-1
using switch activities, 7-2
using while activities, 7-4
conditional processing
with xsl choose, 13-13
with xsl if, 13-12
correlation properties
deployment descriptor, C-4
constant values
in transformations, 13-7
copyList function
description, D-6
core XPath functions
eXamples, 3-3
correlation ID
WS-Addressing, 5-6
correlation sets
associating with receive activities, 5-22
creating, 5-21
creating property aliases, 5-24
definition, 5-7, 5-10
tutorials, 1-11, 5-14
WSDL file content, 5-25
correlation sets tab
in activities, B-2
correlations, 5-7
adding on an OnMessage branch of a pick
activity, B-19
countNodes function, 3-17
definition, 3-7
create instance
definition, 5-5
in receive activities, 5-5
create-delimited-string function
description, D-35
createsInstance attribute, 5-6
create-nodeset-from-delimited-string function
description, D-2
createsWordMLDocument function
description, D-29
current-date function
description, D-18
current-dateTime function description, D-19
current-time function description, D-19
custom sensor publish type, 17-2
custom escalation function using, 15-122
custom plug-ins use with identity service, 15-104, 15-105
custom reporting in BPEL tests, 20-29
custom task display form contents of, 15-74 contents of autogenerated JSP, 15-75 contents of custom JSP, 15-76 contents of default JSP, 15-77 contents of XSL, 15-77
custom XPath functions references, 1-9
customize task activation agent properties, 19-8 deployment descriptor configurations properties, 19-8 example, 19-5 for deploying to multiple environments with different configuration values, 19-5 partnerlinkbinding properties, 19-8 specifying the property value for development and production environments in a single build file location, 19-5 syntax, 19-7

D
Dashboard tab Oracle BPEL Control, 19-13 viewing deployed processes, 15-95
data manipulation accessing fields with complex type variables, 3-6 assigning Boolean values, 3-8 assigning date or time, 3-8 assigning literal strings, 3-7 assigning numeric values, 3-6 concatenating strings, 3-7 converting from a string to a structured XML object type, 3-21 copying data between variables, 3-5 determining array sizes, 3-17 dynamically indexing into a data sequence, 3-17 dynamically indexing with the genEmptyElem function, 3-19 generating array-equivalent functionality with the genEmptyElem function, 3-20 initializing variables, 3-4 manipulating arrays, 3-18 manipulating attributes, 3-9 mathematical calculations with XPath functions, 3-7 statically indexing into a data sequence, 3-16 with assign activities, 3-2, 3-5 with XQuery and XSLT, 3-4
data sequences dynamically indexing into, 3-17 database sensor publish type, 17-2 database adapter advanced tutorial, 1-12 capabilities, B-34 delete operation tutorial, 1-12 fault handling tutorial, 1-12 insert operation tutorial, 1-12 last read ID tutorial, 1-12 logical polling delete strategy, 1-12 master detail replication tutorial, 1-12 merge operation tutorial, 1-12 polling control table tutorial, 1-12 pure SQL tutorial, 1-12 query by example, 1-12 REF_CURSORs tutorial, 1-12 REF_CURSORs tutorial, 1-12 SELECT ALL tutorial, 1-12 SQL Server tutorial, 1-12 stored procedure tutorial, 1-12 tutorials, 1-12, 1-14 update operation tutorial, 1-12 with file adapter tutorial, 1-12 database delete operation tutorial, 1-12 database fault handling tutorial, 1-12 database insert operation tutorial, 1-12 database merge operation tutorial, 1-12 database update operation tutorial, 1-12 database views in BPEL tests, 20-30 dates assigning, 3-8 day-from-datetime function description, D-19 debatching FTP adapter tutorial, 1-14 FTP tutorial, 1-14 tutorial, 1-11, 1-14 decide activity assert facts and execute rule set operation definition, 18-15 assert facts only operation definition, 18-15 assert facts only operation definition, 18-15 assert facts, execute rule set, and retrieve results operation definition, 18-15 assert facts, execute rule set, retrieve results, and reset the session operation definition, 18-15 capabilities, B-5 creating, 18-36 definition, 18-14
execute function and reset the session operation
definition, 18-15
execute function operation definition, 18-15
retrieve results operation definition, 18-15
selecting the operation of the invocation
pattern, 18-15
decision service
advanced features, 18-47
architecture, 18-3
capabilities, B-34
components, 18-4
configuration file contents, 18-5
definition, 18-2
deployment and run time, 18-39, 18-42, 18-43
directory structure, 18-39
interactions with other components, 18-4
JAX-RPC Web services, 18-4, 18-47
logging of session events, 18-48
Oracle BPEL Control support, 18-45
Oracle Enterprise Manager support, 18-43
partner link, 18-39
use case, 18-51
using WSIF bindings, 18-47
Decision Service Builder
definition, 18-4
Decision Service Cache
definition, 18-4
Decision Service Runtime
definition, 18-4
decision service wizard
for integrating business rules with BPEL
processes, 18-9
importing schema files, 18-13
selecting a business rule, 18-11
selecting an invocation pattern, 18-10
specifying input and output facts, 18-12
decision services
customizing assertXpath, 18-50
default
changing the password, 19-10
logging into domains, 19-9
user account, 15-103, 19-9
versions of processes, 19-17
default revision
of BPEL processes, 19-17
defining a fault handler, 8-5
dehydration
definition, 1-2, 5-10
dehydration store, 5-10
deleagations
of tasks, 15-10
demonstrations
location of, 1-6, 1-7
demos
AmazonFlow, 1-7
AutoLoadDemo, 1-7
BankTransferDemo, 1-7
binary attachments in SOAP messages, 1-7, 3-29
binding faults, 8-5, 8-12
BPELTest, 1-7
CheckoutDemo, 1-7
compensate activity, 1-7
DocumentReview, 1-8
ExpenseRequestApproval, 1-8
FlowNActivity, 1-8
GoogleFlow, 1-8
HelpDeskServiceRequest, 1-8
HotwireDemo, 1-8
human workflows, 1-9
IBMSamples, 1-8
Java embedding, 1-8
LoanDemo, 1-8
LoanDemoPlus, 1-8
parallel synchronous invocations, 1-8
ParallelSearch, 1-8
ResilientDemo, 1-8
run-time exceptions, 1-8
run-time faults, 8-5, 8-12
SalesforceFlow, 1-8
sensor actions, 17-3
sensors, 17-3
SleepBroker, 1-8
transformations, 1-9
VacationRequest, 1-9
workflows, 1-8
XSLMapper, 1-9
XML Mapper, 1-9
deploying
VacationRequest process, 15-95
deployment
deploy one process at a time, 19-1
deploy one process at a time of a suitcase JAR file to a specific domain, 19-11
deploy one process at a time of BPEL processes, 19-1
deploy one process at a time of custom Oracle BPEL Worklist Application, 16-46
deploy one process at a time of multiple versions of the same process, 19-2
deploy one process at a time of overwriting versions, 19-2
deploy one process at a time specifying the property value for development and
production environments in a single build file location, 19-5
deploy one process at a time status displays in Log Window, 19-3
deploy one process at a time to multiple environments with different
configuration values, 19-5
deploy one process at a time underdeploying a process from a specific
deploy one process at a time domain, 19-12
deploy one process at a time with Oracle JDeveloper, 19-2
deploy one process at a time with the BPEL Process Deployer, 19-3
deployment descriptor
deploy one process at a time bpel.xmx file, 2-4, C-2
deploy one process at a time configuration properties, C-4, C-5
deploy one process at a time customize task syntax, 19-8
deploy one process at a time defining a configuration property, C-5
deploy one process at a time defining a preference property, C-1
deploy one process at a time encrypting a preference value, C-3
deploy one process at a time getting a preference value within a BPEL
process, C-3
deploy one process at a time preference properties, C-1
deploy one process at a time updating a preference at run time, 19-25, C-3
deploy one process at a time developer prompt
required when deploying services through ant or
obant, 2-3

development mode
changing, 19-11

Diagram window, 2-7
definition, 2-7
location of in Oracle JDeveloper, 2-5
dictionaries
in transformations, 13-19
DIME protocol
SOAP binary attachments, 3-28
doc function
description, D-7
documentation
using this guide, 1-4

DocumentReview
demos, 1-8
domain passwords
changing, 11-3
changing domain passwords, 19-10
domains
best practices, 19-8
changing domain passwords, 19-10
changing domains in Oracle BPEL Control, 19-13
creating, 19-10
creating and managing, 19-8
definitions, 19-2
deploying to specific domains, 19-11
logging into, 19-9
undeploying a process, 19-12
domain.xml file
increasing the syncMaxWaitTime property, 10-4
A-1
location, 10-4
setting properties to catch exception errors, A-1
dynamic assignment
of task participants, 15-26, 15-28, 15-31, 15-34, 15-36
dynamic assignment functions
configuring, 15-117
configuring display names, 15-118
definition, 15-113
implementing, 15-114
dynamic JCA partner links
tutorials, 1-14
dynamic partner links
references, 1-9
dynamic values
setting at run time, 20-27

E

EJB
invoked by a BPEL process through use of WSIF binding, 9-1
security in EJB Web services, 15-98
support in workflow services, 15-97
tutorials, 1-15
EJB Web service
capabilities, B-35
elements
ignoring in XSLT documents, 13-22
e-mail
attachments tutorial, 1-14
dynamically setting addresses, 14-13
making e-mails actionable, 15-46, 15-83
notifications support, 14-2, 14-4
e-mail activation agent
starting processes, 14-14
e-mail activity
capabilities, B-6
e-mail attachments
notifications support, 14-5
tutorial, 1-14
e-mail messages
HTML content for message body, 14-8
empty activity
capabilities, B-8
definition, 8-8
fault handling, 8-8
empty processes
creating an empty process and importing a
schema, A-3
emulation tests
overview, 20-2
emulations
creating, 20-10
emulating BPEL tests, 20-13
emulating faults, 20-12
emulating inbound messages, 20-11
emulating partner tests, 20-13
in BPEL test suites, 20-4
ending
tasks, 15-39
ends-with function
description, D-35
ers
invalid settings, B-37
escalating
tasks, 15-39
escalation policy
escalate after, 15-42
overview, 15-39, 15-40
specifying, 15-47
escalations
of tasks, 15-10
evaluation time
definition, 17-4
event handlers
references, 1-9
exceptions, 8-3
catching, A-1
tutorials, 1-14
execute function and reset the session operation
definition, 18-15
execute function operation
definition, 18-15
Execution Unit
definition, 18-4
ExpenseRequestApproval
demos, 1-8
expiration policy
  expire after, 15-41
  never expire, 15-41
overview, 15-39, 15-40
expires, 15-42
of tasks, 15-10
expression constants
  variable initialization, 3-4
external actions
  verifying in BPEL tests, 20-29
external calls
  creating in BPEL tests, 20-16
external routing service
  configuring, 15-36
  definition, 15-12, 15-36
  specifying the class name, 15-37
  workflow participant type, 15-12, 15-36
facades
  See XML facades
Fact Converter
  definition, 18-4
facts
  specifying input and output facts with the decision service wizard, 18-12
fault handling, 8-5
  binding faults demo, 8-5, 8-12
  defining, 8-5, 8-5, 8-10
  importing RuntimeFault.wsdl, 8-5
  modifying the WSDL files, 8-5
  returning external faults, 8-7
  sample, 1-8
  throwing a WSDL fault from a Java binding, 9-4
  throwing internal faults, 8-6
tutorials, 1-11, 1-14
use case, 8-1
  using catch branches, 8-6
  using compensate activities, 8-9
  using empty activities, 8-8
  using scope activities, 8-6, 8-8
  using terminate activities, 8-10
  using the getFaultAsString function, 8-5
  using throw activities, 8-6
fault sensor reports
  creating, 22-9
fault sensors
definition, 17-2
faults
  categories of faults in BPEL, 8-3
  emulating in BPEL tests, 20-12
  Qname fault name, 8-3
  returning external faults, 8-7
  standard faults, 8-3
  throwing internal faults, 8-6
fax
  notifications support, 14-8
  fax activity
capabilities, B-9
file adapter
capabilities, B-35
COBOL Copybook tutorial, 1-11
complex structure tutorial, 1-11
debatching tutorial, 1-11
flat structure, 1-11
opaque with headers tutorial, 1-11
tutorials, 1-11, 1-14
fire and forget
  one-way message, 12-1
fixed length format
tutorials, 1-11
flat structure
tutorial, 1-11
flow activity
capabilities, B-11
definition, 6-2
  references, 1-9
tutorials, 1-9, 1-10
flowN activity
capabilities, B-12
definition, 6-4
demos, 1-8
  references, 1-9
format function
description, D-41
formatDate function
description, D-8
format-dateTime function
description, D-20
format-string function
description, D-36
FTP adapter
capabilities, B-35
FTP debatching, 1-14
tutorials, 1-14
functions
  abs, D-23
  add-dayTimeDuration-to-dateTime, D-18
  addQuotes, D-5
  appendToList, D-6
  authenticate, D-40
  batchProcessActive, D-40
  batchProcessCompleted, D-40
  chaining in transformations, 13-10
  clearTaskAssignees, D-28
  compare, D-34
  compare-ignore-case, D-34
  concat, 3-7
  copyList, D-6
  countNodes, 3-17, D-7
  create-delimited-string, D-33
  create-nodeset-from-delimited-string, D-2
  createWordMLDocument, D-29
  creating a custom XPath function, D-45
current-date, D-18
current-dateTime, D-19
current-time, D-19
custom, 3-4

Index-10
day-from-dateTime, D-19
deprecated workflow service and identity service
functions, 15-123
descriptions, 13-8
doc, D-7
dynamically setting e-mail addresses and
telephone numbers, 14-13
editing in transformations, 13-9
ingdex-within-string, 13-11
functions prefixed with xp20 or orcl, 13-8
getEmptyElem, 3-20, D-41
generateGUID, D-8
getFirstChildElement, D-41
generate-gui, D-2
getFirstChildString, D-8
get-contract-as-string, D-36
getContentAsString, D-8
getConversationId, D-9
genEmptyElem, 3-20, D-41
generateGUID, D-8
getCreator, D-9
generate-gui, D-2
getDomainId, D-10
generateUID, D-8
getCreator, D-9
generate-gui, D-2
getFaultAsString, 8-5
generateUUID, D-2
getGroupIdsFromGroupAlias, D-11
getGroupProperty, D-24
getTaskAttachmentByIndex, D-30
getGroupProperty, D-24
getTaskAttachmentByName, D-30
getGroupProperty, D-24
getTaskAttachmentContents, D-31
getGroupProperty, D-24
getTaskAttachmentBindingString, D-31
getGroupProperty, D-24
getNumberOfTaskApprovals, D-30
getGroupProperty, D-24
getPreference, D-12
getGroupProperty, D-24
getPreviousTaskApprover, D-30
getGroupProperty, D-24
processXQuery, D-14
getProcessURL, D-12
processXSLT, 14-8, D-14
getProcessVersion, D-13
processXSQL, D-15
getReportees, D-25
query-database, D-17
generateGUID, D-8
readBinaryFromFile, D-15
generateGUID, D-8
readFile, 14-7, D-15
getReference, 1-9
generateGUID, D-8
search, D-43
getReferenceSequence, 1-9
seconds-from-dateTime, D-21
selecting an data sequence element, 3-16
getVariableData, 14-13, D-4
getTimeZone, D-20
sequence-next-val, D-17
generateGUID, D-8
subtract-dayTimeDuration-from-dateTime, D-21
sqrt, D-43
timezone-from-dateTime, D-22
translateFromNative, D-44
translateToNative, D-44
upper-case, D-39
user-defined in transformations, 13-10
workflow related, 15-122
writeBinaryToTree, D-16
year-from-dateTime, D-22
FYI assignee
configuring, 15-35
definition, 15-11, 15-35
workflow participant type, 15-11, 15-35

G
generateGUID function
description, 3-20, D-41
generate-guid function
description, D-2

getChildElement function
description, D-41

getDescriptionAsString function
description, D-8

getCreator function
description, D-9

gerCurrentDate function
description, 3-8, D-9

gerCurrentDateTime function
description, 3-8, D-9

gerCurrentTime function
description, 3-8, D-9

gerDefaultRealmName function
description, D-23

gerDomainId function
description, D-10

gerElement function
description, 3-19, D-10

gerFaultAsString function
description, 8-5

gerGroupIdsFromGroupAlias function
description, D-11

gerGroupProperty function
description, D-24

gerInstanceId function
description, D-11

gerLocalizedString function
description, D-36

gerNotificationProperty function
description, D-29

gerNumberOfTaskApprovals function
description, D-30

gerPreviousTaskApprover function
description, D-12

gerProcessId function
description, D-12

gerProcessOwner function
description, D-12

gerProcessURL function
description, D-12

gerProcessVersion function
description, D-13

gerReportees function
description, D-25

gerTaskAttachmentByIndex function
description, D-30

gerTaskAttachmentByName function
description, D-30

gerTaskAttachmentsCount function
description, D-31

gerTaskResourceId function
description, D-31

gerUserAliasId function
description, D-13

gerUserProperty function
description, D-25

gerUserRoles function
description, D-26

gerUsersInGroup function
description, D-26

gerVariableData function
description, D-7, D-4

gerVariableProperty function
description, D-5

global task variable name
specifying in human task activities, 15-58

GoogleFlow
demos, 1-8

grantees
relationship with roles, 15-104

group information
in worklists, 16-39

group vote
configuring, 15-27
consensus percentage, 15-29
default outcome, 15-29
definition, 15-11, 15-27
immediately triggering a voted outcome when a
minimum percentage is met, 15-29
specifying group voting details, 15-29
waiting until all votes are in before triggering an
outcome, 15-29

worklist participant type, 15-11, 15-27

groups
creating, 15-104
guest
user account, 15-103

H

header variables
viewing details in Oracle BPEL Control, 19-26

headers
bpel:property extension, 3-26
input:HeaderHandler, 3-24
manipulation of partnerLink properties, 3-26
creating users and groups, 15-104
definition, 15-4, 15-6, 15-102
deprecated functions, 15-123
determining a user’s local language and time zone, 16-39
EJB, SOAP, and Java support, 15-97, 16-57
functions
getDefaultRealmName, D-23
getGroupProperty, D-24
getManager, D-24
getReportees, D-25
getUserProperty, D-25
getUserRoles, D-26
lookupUser, D-28
isUserInRole, D-27
lookupGroup, D-27
multirealm support, 15-107
providers, 15-104, 15-105
support for in workflows, 15-4, 15-102
supported task operations, 15-102
use with custom plug-ins, 15-104, 15-105
use with JAZN, 15-4, 15-102, 15-104, 15-105
use with LDAP, 15-4, 15-102, 15-104
use with third-party LDAP servers, 15-105
user role properties, 15-106
WSDL file location, 15-98
XPath extension functions, 15-122
implicit-timezone function
description, D-20
import
an existing BPEL process into a project, 2-4
cannot import XSD files in a ZIP file, 2-4
of schemas into an empty BPEL process, A-3
use with LDAP, 15-4, 15-102, 15-104
use with third-party LDAP servers, 15-105
user and role properties, 15-106
WSDL file location, 15-98
XPath extension functions, 15-122
index-within-string function
description, D-37
indexing methods
using XPath, 3-17
index-within-string function
description, D-37
inspection.xml file
adding remote Web services, 4-4
instances
starting new, 5-6
Instances tab
Oracle BPEL Control, 19-26
timeout, 12-4
index-within-string function
description, D-37
integration server connection
using to deploy a process, 15-95
interaction patterns
asynchronous interaction with notification timer, 12-9
asynchronous interaction with timeout, 12-4
asynchronous interactions, 12-3
common patterns between a BPEL process and another application, 12-1
multiple interactions, 12-10
one request, a mandatory response, and an optional response, 12-8
one request, multiple responses, 12-6
one request, one of two possible responses, 12-7
one-way message, 12-1
partial processing, 12-9
synchronous interactions, 12-2
interop
location of, 1-7
interoperability
with Apache Axis, 1-7
with BEA WebLogic, 1-7
with Microsoft Net, 1-7
Invalid Settings error message, B-37
invocation patterns
assert facts and execute rule set operation definition, 18-15
assert facts only operation definition, 18-15
assert facts, execute rule set, and retrieve results operation definition, 18-15
assert facts, execute rule set, retrieve results, and reset the session operation definition, 18-15
execute function and reset the session operation definition, 18-15
execute function operation definition, 18-15
retrieve results operation definition, 18-15
selecting the operation in the decide activity, 18-15
selecting with the decision service wizard, 18-10
supported, 18-10
invoke activity, 5-5
adding to an asynchronous service, 5-12
capabilities, B-14
definition, 4-2
in asynchronous services, 5-5, 5-12
in synchronous services, 4-2, 4-5
references, 1-9
tutorials, 1-10
isUserInRole function
description, D-27
Java
J2EE security
supported, 19-9
JAAS security
supported, 19-9
Java
support in workflow services, 15-97
Java API
invoking a BPEL process with the generic Java API, 11-2
invoking a one-way operation, 11-4
invoking a two-way operation, 11-3
NormalizedMessage class for dynamically activating messages, 11-3
J
Index-15

parsing XML messages, 11-3
using from a remote client, 11-5
using the Locator class to connect to Oracle BPEL Process Manager, 11-2
Java applications
wrapped as SOAP services, 9-5
Java embedding
BPEL-compatible interface, 9-1
bpelx:exec extension, 9-1
demos, 1-8
embedding code in a BPEL process, 9-6
example, 9-8
in a BPEL process, 9-1
invoking an EJB, 9-1
tutorials, 9-5
using bpelx:exec, 9-6
using WSDL binding, 9-1
Java embedding activity
capabilities, B-16
Java method
tutorials, 1-15
Java RMI client
security tutorial, 1-14
Java Web service
capabilities, B-35
Javadoc
deployment, 11-3, 15-106
JavaExec
references, 1-9
Java/JSF applications
calling a BPEL process, 11-1
JAX-RPC
tutorials, 1-10
JAX-RPC Web services
decision service run time, 18-4, 18-47
JAZN
definition, 15-105
LDAP-based (Oracle Internet Directory) provider type, 15-105
storing a user’s local language and time zone, 16-39
use with identity service, 15-4, 15-102, 15-104, 15-105
use with third-party LDAP servers, 15-105
use with XML-based provider type, 15-105
jazn-data.xml file
creating users and groups with XML-based JAZN provider, 15-104
JBoss
supported by Oracle BPEL Process Manager, 1-2
JMS adapter
capabilities, B-35
tutorials, 1-13
JMS Queue
sensor publish type, 17-2
JMS Topic
c sensor publish type, 17-2
JSP client
security tutorial, 1-14
JSP form
tsamples, 1-7
tutorials, 1-10
JUnit
BPEL test reports, 20-3
BPEL test results, 20-21, 20-23
definition, 20-3
languages
accessing the Oracle BPEL Worklist Application in local languages, 16-39
setting in JAZN, 16-39
setting in LDAP, 16-39
last read ID
tutorial, 1-12
last-index-within-string function
description, D-37
LDAP
storing a user’s local language and time zone, 16-39
used with identity service, 15-4, 15-102, 15-104
left-trim function
description, D-38
life cycles
default instance version, 19-20, 19-22
default revision, 19-17
example, 19-19
inactive instances, 19-23
of processes, 19-14, 19-17
retiring instances, 19-22
undeployed, 19-17
links
calls, 1-9
listUsers function
description, D-42
literal strings
assigning, 3-7
literal XML
variable initialization, 3-4
LoanDemo
demos, 1-8
LoanDemoPlus
demos, 1-8
Log window
definition, 2-13
location of in Oracle JDeveloper, 2-5
logging
of business rules session events, 18-48
logging in
Oracle BPEL Worklist Application, 16-4
logical polling delete strategy
tutorial, 1-12
logs
reviewing process logs from Oracle BPEL Control, 19-15
lookup-dvm
Index-16

not supported on Oracle BPEL Process Manager, D-2
lookup-dvm function
description, D-2
lookupGroup function
description, D-27
lookup-table function
description, D-16
lookupUser function
description, D-28
lookup-xml function
description, D-3
lower-case function
description, D-38

M
management chain
configuring, 15-30
definition, 15-11, 15-30
highest title of approver, 15-32
maximum number of chain levels up, 15-32
workflow participant type, 15-11, 15-30
map parameters
creating in transformations, 13-19
map variables
creating in transformations, 13-19
master detail replication
tutorial, 1-12
matches function
description, D-39
maxOccurs attribute, 3-16, 3-17
setting for transformations, 13-27
max-value-among-nodeset function
description, D-42
message flow
tutorials, 1-11
message header handling, 3-24
messages
recovering from Oracle BPEL Control, 19-15
Microsoft .Net sample of interoperability with Oracle BPEL Process Manager, 1-7
MIME protocol
SOAP binary attachments, 3-28
setOccurs attribute
setting for transformations, 13-28
minutes-from-dateTime function
description, D-21
min-value-among-nodeset function
description, D-43
modes
changing for the Oracle BPEL Server, 19-11
month-from-dateTime function
description, D-21
MQ adapter
capabilities, B-36
multilingual settings
specifying in tasks, 15-48, 15-83
multi realms
identity service support for, 15-107
supported, 21-3
myRole attribute
definition, 5-5

N
named templates
creating, 13-10
in functions, 13-10
naming conventions
for BPEL projects, 2-4
NLS
configuration overview, 15-124
nonBlockingInvoke property
using in bpel.xml file, 1-8
NormalizedMessage class
for dynamically activating messages, 11-3
specifying the address of a Web service for the callback, 11-5
viewing the specific field name for the conversation ID, 11-5
notification messages
editing, 15-45
notification services
actionable e-mails, 15-83
configuring the notification channel, 15-81
definition, 15-4, 15-6, 15-108
limitations on setting validateXML to true, 14-15
multilingual settings, 15-83
notification content, 15-82
reliability support, 15-84
sending inbound and outbound attachments, 15-84
sending inbound comments, 15-84
sending reminders, 15-85
sending secure notifications, 15-85
specifying participant notification preferences, 15-43
WSIL file location, 15-98
notifications
configuring in Oracle JDeveloper, 14-3
definition, 2-19, 14-1, 15-2
dynamically setting e-mail addresses and telephone numbers, 14-13
e-mail attachment support, 14-5
e-mail attachment tutorials, 1-14
e-mail support, 14-2, 14-4
fax support, 14-4
formatting the e-mail message body as HTML, 14-8
in Oracle JDeveloper, 2-19
number of retries, 14-3
number of retry intervals, 14-3
page support, 14-10
reliable notification service, 14-2
selecting recipients by browsing the user directory, 14-14
SMS support, 14-11
tutorials, 1-14
use case, 14-1
using Oracle Application Server Wireless, 14-2
voice mail support, 14-12
notifications and reminders in tasks, 15-80
numeric values assigning, 3-6

O
obant
using the developer prompt, 2-3
oc4jadmin
changing the password, 19-10
logging into domains, 19-9
making the password compatible with the samples, 1-4
user account, 15-103, 19-9
off
process state, 19-17
on
process state, 19-17
onAlarm branch of pick activity, 10-2
onMessage branch of pick activity, 10-2
opaque with headers tutorial, 1-11
Oracle Application Server Integration B2B
integration with Oracle BPEL Process Manager, 1-3
Oracle Application Server Integration InterConnect
integration with Oracle BPEL Process Manager, 1-3
Oracle Application Server Portal
accessing Oracle BPEL Control reports, 21-2, 21-9
accessing Oracle BPEL Worklist Application, 21-2, 21-13
activity sensor report type portal, 21-13
analysis portlet view, 21-19
configuring realms for 10.1.3.1.0, 21-3
deploying portlets, 21-4
fault sensor report type portal, 21-13
installing for use with Oracle BPEL Process Manager, 21-3
instance execution time report type portal, 21-11
instance state report type portal, 21-11
listing portlet view, 21-16
Oracle BPEL Worklist Application parameters, 21-14, 21-17
overview, 21-1
performance report type portal, 21-12
registering Web providers, 21-6
viewing Oracle BPEL Control reports from, 22-3
Oracle Application Server Wireless
wireless and voice component, 14-2
Oracle Applications adapter
capabilities, 8-36
changing a purchase order tutorial, 1-14
demonstrating outbound business events tutorial, 1-14
importing and creating a purchase order in Oracle E-Business suite tutorial, 1-14
integration with Oracle XML gateway to create a purchase order tutorial, 1-14
integration with Oracle XML gateway to retrieve a purchase order acknowledgement tutorial, 1-14
Oracle BAM
See Oracle Business Activity Monitoring
Oracle BAM Server
creating a BPEL sensor, 17-15
creating a BPEL sensor action, 17-15
creating a connection to, 17-14
deployment descriptor parameters, 17-15
Oracle BPM Admin Console
accessing, 19-10
supported Web browsers, 19-10
Oracle BPEL Control
accessing, 15-95
accessing through Oracle Application Server Portal, 21-2, 21-9
Activities tab, 19-27
activity sensor report creation, 22-7
activity sensor report portal, 21-13
BPEL Processes tab, 19-14
changes that impact the worklist application, 15-80
changing to different domains, 19-13
clearing the WSDL cache, 19-14
creating test cases, 20-17
dashboard tab, 19-13
decision service support, 18-45
deploying BPEL processes, 19-14
fault sensor report creation, 22-9
fault sensor report portal, 21-13
instance execution time report portal, 21-11
instance state report portal, 21-11
Instances tab, 19-26
location of function descriptions, 3-4
message recovery from Oracle BPEL Control, 19-15
overview, 2-17
performance report creation, 22-5
performance report portal, 21-12
process report creation, 22-3
process time distribution report creation, 22-10
refreshing the alarm table, 19-15
report creation, 19-25, 22-1
reviewing process logs, 19-15
running a process, 15-95
running and managing processes from, 2-1
setting deployment descriptors, 19-25
supported Web browsers, 19-10
undeploying processes, 19-18
viewing a history of process activities, 19-26
viewing activity interactions, 19-26
viewing and managing processes, 19-12
viewing header variable details, 19-26
viewing sensor and sensor action metadata, 17-11
viewing sensor data, 19-25
viewing sensor values, 19-26
viewing the audit trail of a process, 19-26
Oracle BPEL Process Manager
adds value and ease of use to BPEL functionality, 1-2
components of, 2-1
definition, 1-2
integration with Oracle Application Server Integration B2B, 1-3
integration with Oracle Application Server Integration InterConnect, 1-3
provides support for, 1-2
services in Oracle JDeveloper, 2-18
supported designer types, 1-3
Oracle BPEL Server
changing modes, 19-11
deploying process to, 2-1
overview, 2-17
Oracle BPEL Worklist Application
accessing, 15-96
accessing in local languages, 16-39
accessing tasks assigned to users, 15-10
accessing through Oracle Application Server Portal, 21-2, 21-13
adding comments and attachments, 16-13
analysis portlet view, 21-19
approving or rejecting a task, 15-96
architecture, 16-41
customizing, 16-41
customizing with preferences, 16-48
defining portlet parameters through Oracle Application Server Portal, 21-14, 21-17
definition, 15-4
deploying the custom application, 16-46
determining action permissions, 16-17
enabling for single sign-on, 16-50
home page contents, 16-6
listing portlet view, 21-16
logging in, 16-4
parallel tasks, 16-17
payload viewing, 16-13
reports, 16-33
request status, 16-11
requesting more information, 16-15
responding to tasks from, 15-2
routing, 16-15
sample, 1-8
Task Actions list, 16-10
task reassignment, 16-15
use cases, 16-1
user and group information, 16-39
Oracle Business Activity Monitoring
creating a BPEL sensor action for Oracle BAM Server, 17-15
creating a BPEL sensor for Oracle BAM Server, 17-15
creating a connection to Oracle BAM Server, 17-14
definition
integration with Oracle BPEL Process Manager sensors, 17-13
message batching limitations, 17-17
message batching limitations, 17-17
Oracle Business Rules
use Oracle JAXB 1.0 for XML fact types, 18-22
Oracle Business Rules RL Language
definition, 18-3
Oracle Business Rules Rule Author
definition, 18-3
Oracle Business Rules Rules Engine
definition, 18-3
Oracle Business Rules SDK
definition, 18-3
Oracle Delegated Administration Services
creating users and groups, 15-104
Oracle Enterprise Manager
decision service support, 18-43
Oracle Internet Directory
creating users and groups, 15-104
LDAP-based provider type, 15-105
object classes for specifying user and role properties, 15-106
storing a user’s local language and time zone, 16-39
Oracle JDeveloper
adapters, 2-20
Application Navigator, 2-6
best practices when editing project files, 2-14
compiling and deploying a process, 19-2
compiling without deploying, 19-4
Component Palette, 2-10, 2-11
customizing notifications, 14-3
creating a project, 2-3
creating an application, 2-3
creating sensors, 17-2
deploying test suites, 20-19
designing processes, 2-1
designing BPEL processes, 1-3
location of Application Navigator, 2-5
location of Component Palette, 2-5
location of Diagram window, 2-5
location of Log window, 2-5
location of Process Activities, 2-5
location of Property Inspector, 2-5
location of Structure window, 2-5
Log window, 2-13
notifications, 2-19
overview of design environment, 2-3
overview of design environment, 2-3
overview of services, 2-18
Process Activities, 2-10
Property Inspector, 2-12
servers, 2-19
Services, 2-11
Structure window, 2-12
transformations, 2-19, 13-1
workflows, 2-19
worklists, 2-19
Oracle Rules Engine File Repository
definition, 18-3
Oracle Rules Engine WebDav Repository
definition, 18-3
OracleAS Portal
See Oracle Application Server Portal
orion.ejb-jar.xml file
setting properties to catch exception errors, A-1
outcome-based modeling, 15-64
case statements for other task conclusions, 15-65
payload updates, 15-65
overview, 17-1

P

customize task syntax, 19-8
partnerLinkTypes
definition, 4-3, 5-3
in asynchronous services, 5-3
in synchronous services, 4-3
partnerRole attribute
definition, 5-5
passwords
changing domain passwords, 19-10
for using samples, 1-6
patterns of interaction between a BPEL process and another application, 12-1
payload viewing in worklists, 16-13
performance reports
creating, 22-5
Perl
tutorials, 1-10
PHP
tutorials, 1-10
pick activity
adding correlations on an OnMessage branch, B-19
capabilities, B-19
code example, 10-3
condition branches, 10-2
for timeouts, 10-1
onAlarm branch, 10-2
onMessage branch, 10-2
references, 1-9
tutorial, 1-11
polling control table
tutorial, 1-12
port types
definition, 4-3
in asynchronous services, 5-3
in synchronous services, 4-3
portlet
overview, 21-1
portlet parameter
overview, 21-1
ports
in synchronous services, 4-2
position function
description, 5-17
post() method
support for client specifying a conversation ID, 11-5
preference properties
deployment descriptor, C-1
Oracle BAM Server deployment descriptors, 17-15
SLACompletionTime deployment descriptor, 22-5
Process Activities
definition, 2-10
location of in Oracle JDeveloper, 2-5
process code coverage tests
overview, 20-3
process history
viewing, 19-26
process initiation
in BPEL test suites, 20-4
process life cycles
default instance version, 19-20, 19-22
example, 19-19
inactive instances, 19-23
retiring instances, 19-22
process reports
creating, 22-3
process time distribution reports
creating, 22-10
processes
creating templates of BPEL processes, 2-4
deploying from Oracle BPEL Control, 19-14
naming conventions, 2-4
process life cycle recommendations for a
development environment, 19-18
process life cycle recommendations for a
production environment, 19-18
viewing deployed and completed
processes, 19-15
viewing logs from Oracle BPEL Control, 19-15
processXQuery function
description, D-14
processXSLT function
description, D-14
tutorial, 1-11
processXSQL function
description, D-15
procurement process
workflow functionality example, 15-5
production mode
changing, 19-11
projects
best practices when editing, 2-14
blank spaces not permitted in directory path, 2-3
BPEL file, 2-6
bpel.xmi, 2-6
creating, 15-89
creating in Oracle JDeveloper, 2-3
creating templates of BPEL processes, 2-4
definition, 2-3
importing two schema files of the same name into
the same project is not supported, 2-13
in Application Navigator, 2-6
naming conventions, 2-4
naming limitations, 2-3
WSDL file, 2-6
property aliases
creating for correlation sets, 5-24
Property Inspector
definition, 2-12
location of in Oracle JDeveloper, 2-5
proxy support
in workflow services, 15-10
PUBLIC role
identity service role, 15-103
public views
BPEL_ACTIVITY_SENSOR_VALUES, 17-19
BPEL_FAULT_SENSOR_VALUES, 17-20
BPEL_PROCESS_INSTANCES, 17-18
BPEL_VARIABLE_SENSOR_VALUES, 17-21
BPEMERRORS, 17-22
sensors, 17-18
publish types
creating a custom publisher, 17-8
custom, 17-2
database, 17-2
definition, 17-2
JMS Adapter, 17-2
JMS Queue, 17-2
JMS Topic, 17-2
pure SQL
tutorial, 1-12
Q
Qname
fault name, 8-3
query by example
tutorial, 1-12
query-database function
description, D-17
R
readBinaryFromFile function
description, D-15
readFile function
description, D-15
eexample, 14-7
reading files from absolute directory paths, D-16
realms
support for multirealms, 15-107, 21-3
receive activity
adding to an asynchronous service, 5-13
associating with correlation sets, 5-22
capabilities, B-20
create instance, 5-5
creating new instances, 5-6
in asynchronous services, 5-5, 5-13
tutorials, 1-10
recovery
recovering messages from Oracle BPEL
Control, 19-15
REF CURSORS
tutorial, 1-12
references
assign activity, 1-9
BPEL exec extension, 1-9
BPELTest, 1-9
catch branch, 1-9
custom XPath functions, 1-9
dynamic partner links, 1-9
event handlers, 1-9
flow activity, 1-9
flowN activity, 1-9
functions, 1-9
invoke activity, 1-9
JavaExec, 1-9
links, 1-9
location of, 1-7
pick activity, 1-9
receive activity, 1-9
replay, 1-9
reply activity, 1-9
switch activity, 1-10
terminate activity, 1-10
throw activity, 1-10
while activity, 1-10
XPath functions, 1-10
XPath queries, 1-10
rejectedMessageHandler property
tutorial, 1-13
reliable notification service
persisting messages, 14-2
reminders
for task notifications, 15-85
remoteFault
definition, 8-5
renewing
tasks, 15-39
repeating elements
in transformations, 13-14
replay
references, 1-9
replayFault
definition, 8-5
reply activity
capabilities, B-21
definition, 5-10
in asynchronous services, 5-10
references, 1-9
reporting schema
for database publish type of sensors, 17-18
reports
activity sensor report creation, 22-7
correcting memory errors when generating for transformations, 13-27
creating, 19-25, 22-1
customizing sample XML generation for transformations, 13-27
exporting to comma-separated value (CSV) files, 22-3
fault sensor report creation, 22-9
generating for transformations, 13-26
overview, 22-1
performance report creation, 22-5
process report creation, 22-3
process time distribution report creation, 22-10
tutorials, 1-14
viewing from Oracle Application Server Portal, 22-3
Worklist Application, 16-33
request status
in worklists, 16-11
requesting more information in worklists, 16-15
ResilientDemo
demos, 1-8
resource bundles
for displaying tasks in different languages, 15-48, 15-83
REST services
tutorials, 1-15
retired
process life cycle, 19-17
retrieve results operation
definition, 18-15
retryInterval property
defining for fault handling, 8-12
retryMaxCount property
defining fault handling, 8-12
right-trim function
description, D-39
RMI
tutorials, 1-10
using Java API through, 11-5
roles
BPMDefaultDomainAdmin, 15-103, 15-104
BPMSystemAdmin, 15-103, 15-104
BPMWorkflowAdmin, 15-103, 15-104
BPMWorkflowReassign, 15-103, 15-104
BPMWorkflowSuspend, 15-103, 15-104
BPMWorkflowViewHistory, 15-103, 15-104
for partner links in asynchronous services, 5-4
predefined, 15-103
properties, 15-106
PUBLIC, 15-103
relationship with grantees, 15-104
routing
in worklists, 16-15
route slip
definition, 15-3, 15-26
RPC styles
differences with document-literal styles in WSDL files, 3-2, 3-22
Rule Provider Interface
definition, 18-4
rule set modeling
methodology, 18-17
rules engine
connecting to, 18-7
definition, 18-2
rules engines
supported, 18-8
running
OrderApproval, 15-95
runtime config service
definition, 15-6
EJB, SOAP, and Java support, 15-97, 16-57
supported task operations, 15-111
WSDL file location, 15-98
run-time exceptions, 8-3
demos, 1-8
run-time faults
  binding faults sample, 8-5, 8-12
definition, 8-3
example, 8-5
RuntimeFault.wsdl file
  importing into a process, 8-5

S
  SAASim
    implementation of, 3-33
    samples
      AmazonFlow, 1-7
      Apache Axis, 1-7
      asynchronous services, 1-8
      AutoLoanDemo, 1-7
      automatically creating, 1-6
      BankTransferDemo, 1-7
      BEA WebLogic, 1-7
      BPELTest, 1-7
      CheckoutDemo, 1-7
      DocumentReview, 1-8
      ExpenseRequestApproval, 1-8
      fault handling, 1-8
      flowN activity, 1-8
      GoogleFlow, 1-8
      HelpDeskServiceRequest, 1-8
      HotwireDemo, 1-8
      human workflows, 1-9
      IBMSamples, 1-8
      Java embeddding, 1-8
      JSP, 1-7
      JTA transaction management, 1-7
      LoanDemo, 1-8
      LoanDemoPlus, 1-8
      location of, 1-6, 1-7
      Microsoft.Net, 1-7
      nonBlockingInvoke property in bpel.xml, 1-8
      Oracle BPEL Worklist Application, 1-8
      Parallel Search, 1-8
      parallel workflow, 1-8
      ParallelSearch, 1-8
      password to use, 1-6
      ResilientDemo, 1-8
      run-time exceptions, 1-8
      SalesforceFlow, 1-8
      SleepBroker, 1-8
      transaction management, 1-7
      VacationRequest, 1-9
      wait activity, 1-8
    workflow services, 1-8
    XSLMapper, 1-9
  schema files
    creating a transformation map file from imported schemas, 13-3
    importing into the decision service wizard, 18-13
    replacing in the XSLT Mapper, 13-23
    definition, 19-29
    examples, 19-30
    generating XML facades from WSDL or XSD files, 9-7, 19-29
    using, 19-29
  scope activity
    capabilities, B-22
definition, 8-6
    fault handling, 8-6, 8-8
tutorials, 1-9
  scope name
    specifying in human task activities, 15-38
  search function
definition, D-43
  seconds-from-dateTime function
definition, D-21
  security
    invoking from Java RMI client, 1-14
    invoking from JSP client, 1-14
    invoking over HTTP directly, 1-14
    invoking over SOAP client, 1-14
    J2EE and JAAS security features, 19-9
tutorials, 1-14
  security model
    for workflow services, 15-98
    in EJB Web services, 15-98
    in SOAP Web services, 15-98
  workflow context on behalf of a user, 15-99
  SELECT ALL
  tutorial, 1-12
  sensor actions
    configuring, 17-6
    creating a BPEL sensor action for Oracle BAM Server monitoring, 17-15
demos, 17-3
    registering in the bpel.xml file, 17-11
    viewing definitions, 17-11
    viewing metadata from Oracle BPEL Control, 17-11
    XSD schema file, 17-23
  sensor data
    persisting in a reporting schema, 17-18
  sensor values
    viewing, 19-26
    sensorAction.xml file, 17-6, 17-7
    sensors, 17-1
    activity sensors, 17-1
    BPEL reporting schema, 17-18
    BPEL_ACTIVITYSENSOR_VALUES public views, 17-19
    BPEL_FAULTSENSOR_VALUES public views, 17-20
    BPEL_PROCESS_INSTANCES public views, 17-18
    BPEL_VARIABLESENSORVALUES public views, 17-21
    BPMERRORS public views, 17-22
    configuring, 17-3
    creating a BPEL sensor for Oracle BAM Server to monitor, 17-15
    creating a connection to Oracle BAM Server, 17-14
SSO  
See single sign-on

stored procedures tutorial, 1-12

strings  
concatenating, 3-7  
converting to an XML element, 3-21  
location of in Oracle JDeveloper, 2-5

style sheets  
using for attachments, 15-48

subtract-dayTimeDuration-from-dateTime function description, D-21

suitcase JAR file  
definition, 19-5

tutorial, 1-12

strings  
concatenating, 3-7

System window  
definition, 2-12

location of in Oracle JDeveloper, 2-5

switch activity  
capabilities, B-27

in conditional branching logic, 7-2

required ports, 4-2

syncMaxWaitTime property, 4-2

synchronous interactions

BPEL process as the client, 12-3

BPEL process as the service, 12-3

definition, 12-2

returning faults, 8-7

synchronous processes

timeouts, 10-4

synchronous reads

FTP tutorial, 1-14

synchronous services

callbacks with the partner link and invoke activity, 4-2

calling, 4-5

invoke activities, 4-5

partnerLinkTypes, 4-3

ports, 4-3

ports, 4-2

tutorials, 1-10, 1-14

syncMaxWaitTime property

in synchronous callbacks, 4-2

increasing to prevent timeouts, 10-4

setting to catch exception errors, A-1

in tasks, 15-49

T

task action time limits

specifying, 15-26, 15-29, 15-32, 15-34

task actions

in worklists, 16-3, 16-10

task approvers

specifying statically or dynamically, 15-23

specifying the number of, 15-32

task assignment and routing

definition, 15-3

task conditions

abruptly completing a condition, 15-38

task display form

adding update support to custom JSPs, 15-80

changes between 10.1.2 and 10.1.3.1.0, E-8

contents of custom task display form, 15-74

contents of custom task display form automatically generated JSP, 15-75

contents of custom task display form custom JSP, 15-76

contents of custom task display form default JSP, 15-77

creating custom JSPs, 15-78

custom task display form, 15-74

custom task display form automatically generated JSP, 15-75, 15-77

custom task display form custom JSP, 15-76

custom task display form XSL, 15-77

customizing the simple task display form default JSP, 15-72

customizing the simple task display form default JSP to display a check box, 15-74

customizing the simple task display form JSP, 15-71

customizing the simple task display form map file, 15-71

definition, 15-13, 15-66

deleting, 15-78

deploying, 15-77

payload of simple task display form, 15-68

preview release support for ADF data controls, 15-67

selecting, 15-66

simple task display form, 15-68

definition, 15-12, 15-33

simple task display form payload, 15-68

task file

associating with a BPEL process, 15-12, 15-33, 15-54

definition, 15-3, 15-12, 15-13

eventing the file and saving changes, 15-53

task history

specifying in human task activities, 15-59, 15-64

task initiator

specifying, 15-55

task metadata service

definition, 15-6

EJB, SOAP, and Java support, 15-97, 16-37

supported task operations, 15-109

WSDL file location, 15-97

task notification

editing notification messages, 15-45

making e-mails actionable, 15-46, 15-83

notifying recipients of changes to task status, 15-44

overview, 15-43
reminders, 15-85
securing notifications, 15-46, 15-85
setting up reminders, 15-46, 15-84
task attachments with e-mail notifications, 15-46, 15-84

task outcome
displaying custom outcomes in a human task activity, 15-17
specifying, 15-16
task owner
specifying by browsing the user directory, 15-18
specifying in human task activities, 15-58
specifying through XPath expressions, 15-20

task parameters
specifying, 15-56
task participants
allowing all participants to invite other participants, 15-37
assigning task participants by name or expression, 15-25, 15-28, 15-31, 15-33, 15-36
bypassing, 15-26, 15-28, 15-31, 15-34
dynamically assigning task participants by expression, 15-26, 15-28, 15-31, 15-34, 15-36
dynamically assigning with the assignment service, 15-118
inviting additional task participants, 15-26, 15-32, 15-34
sharing attachments and comments, 15-29, 15-36
specifying, 15-22
task payload data structure
specifying, 15-21
task priority
specifying, 15-16, 15-55
task query service
definition, 15-6
EJB, SOAP, and Java support, 15-97, 16-57
supported task operations, 15-101
using over SOAP, 15-102
WSIDL file location, 15-97
task reassignment
in worklists, 16-15
task reminders
setting up, 15-46
task reports service
EJB, SOAP, and Java support, 15-97, 16-57
task routing service
definition, 15-6
task service
definition, 15-6
EJB, SOAP, and Java support, 15-97, 16-57
supported task operations, 15-99
WSIDL file location, 15-97
task title
specifying, 15-16, 15-55
tasks
assigning a task to a user or role, 15-9
continuing workflows from other workflows, 15-12
definitions, 15-2
delegations, 15-10
escalating, renewing, or ending a task, 15-39
escalations, 15-10
expirations, 15-10
notifications and reminders, 15-80
overriding exception management, 15-51
routing, 15-9
TCP tunneling
definition, 5-8
setting up a TCP listener for asynchronous services, 5-9
setting up a TCP listener for synchronous services, 5-8
technology adapters
supported, 1-2
templates
creating templates of BPEL processes, 2-4
terminate activity
abilities, B-27
definition, 8-10
fault handling, 8-10
references, 1-10
test cases
creating from Oracle BPEL Control, 20-17
creating in test suites, 20-9
editing, 20-9
importing into test suites, 20-7
test suits
components, 20-3
creating, 20-6
definition, 20-2
deploying from an ant task, 20-20
deploying from Oracle JDeveloper, 20-19
limitations on multibyte character names, 20-6
running from an ant task, 20-24
running from Oracle BPEL Control, 19-25, 20-21
viewing test results, 20-21
third-party adapters
supported, 1-2
throw activity
abilities, B-28
references, 1-10
throwing internal faults, 8-6
time
assigning, 3-8
time duration format, 10-2
timeouts
designing, 10-4
increasing the syncMaxWaitTime property, 10-4
of BPEL processes, 10-1
tutorial, 1-11
use case, 10-1
using pick activities, 10-1
using the wait activity, 10-4
with synchronous processes, 10-4
timezone-from-dateTime function
description, D-22
timezone-from-time function
description, D-22
transaction management
samples, 1-7
transaction-manager.xml file
handling long-running processes, A-2
VacationRequest process, 15-95
validation
limitations on setting to true for notification services, 14-15
of XML data with bpelx validate, 3-16
when loading a process diagram, B-37
validating
correcting in complex processes, A-2
variable sensors
definition, 17-2
variables
complex type, 3-6
copying data between, 3-5
initializing with expression constants, 3-4
initializing with literal XML, 3-4
versions
default, 19-17
definition, 19-2
deployment of multiple versions of the same process, 19-2
do not overwrite existing versions of a process with newer versions in a production environment, 19-19
in production and development environments, 19-3
deployment of multiple versions of the same
life cycles, 19-14
of BPEL processes, 19-2
overwriting, 19-2
voice activity
capabilities, B-30
voice mail
dynamically setting telephone numbers, 14-13
notifications support, 14-12

in production and development environments, 19-3
life cycles, 19-14
of BPEL processes, 19-2
overwriting, 19-2
voice activity
capabilities, B-30
voice mail
dynamically setting telephone numbers, 14-13
notifications support, 14-12

W
wait activity
capabilities, B-31
code example, 10-4
definition, 10-4
references, 1-10
sample, 1-8
Web browsers
supported, 19-10
Web client
tutorials, 1-10
Web interfaces
interacting with BPEL processes, 11-1
Web providers
overview, 21-1
registering with Oracle Application Server Portal, 21-6
Web services
remote Web Services
selecting remote services when creating a partner link, 4-4
selecting remote services when creating a partner link, 4-4
Web Service/SOAP interface

invoking a BPEL process, 11-6
wfeDynamicGroupAssign function
description, D-32
wfeDynamicUserAssign function
description, D-33
while activity
capabilities, B-32
in conditional branching logic, 7-4
references, 1-10
tutorial, 1-11
WordML style sheets
using for attachments, 15-48
work queues
in workflow services, 15-10
workflow context
creating on behalf of a user, 15-99
workflows
overview, 15-97
workflow services
abruptly completing a condition, 15-38
actionable e-mails, 15-46, 15-83
adding update support to custom JSPs, 15-80
allowing all participants to invite other participants, 15-37
assigning a task to a user or role, 15-9
assigning task participants by name or expression, 15-25, 15-28, 15-31, 15-33, 15-36
assignment service configuration, 15-113
associating the human task activity with a BPEL process, 15-54
associating the human task definition with a BPEL process, 15-12
associating the human task with a BPEL process, 15-53
automatic assignment and delegation, 15-10
bypassing task participants, 15-26, 15-28, 15-31, 15-34
changes between 10.1.2 and 10.1.3.1.0, 15-2, E-6
changes between the workflow wizard and Human Task editor between 10.1.2 and 10.1.3.1.0, E-2
changes to configuration files between 10.1.2 and 10.1.3.1.0, E-6
changes to task display form deployment between 10.1.2 and 10.1.3.1.0, E-8
changes to workflow APIs between 10.1.2 and 10.1.3.1.0, E-7
components of, 15-6
continuing workflows from other workflows, 15-12
creating custom JSPs, 15-78
custom task display form, 15-74
custom task display form autogenerated JSP, 15-75
custom task display form custom JSP, 15-76
custom task display form default JSP, 15-77
customizing the simple task display form default JSP, 15-72
customizing the simple task display form default
JSP to display a check box, 15-74
customizing the simple task display form JSP, 15-71
customizing the simple task display form map file, 15-71
definition, 2-19, 15-2
deleting the task display form, 15-78
demos, 1-8
deploying the task display form, 15-77
deprecated functions, 15-123
dynamically assigning task participants by expression, 15-26, 15-28, 15-31, 15-34, 15-36
editing notification messages, 15-45
EJB support, 15-97
escalate after policy, 15-42
escalating, renewing, or ending a task, 15-39
escalation and expiration policy overview, 15-39, 15-40
escalation rules, 15-47
escalation, expiration, and delegation, 15-10
exiting the Human Task editor and saving changes, 15-53
expire after policy, 15-41
external routing service task participant, 15-36
features, 15-2
functions, 15-122
clearTaskAssignees, D-28
createWordMLDocument, D-29
getNotificationProperty, D-29
getNumberOfTaskApprovals, D-30
getPreviousTaskAppraiser, D-30
getTaskAttachmentByIndex, D-30
getTaskAttachmentByName, D-30
getTaskAttachmentContents, D-31
getTaskAttachmentCount, D-31
getTaskResourceBindingString, D-31
wfDynamicGroupAssign, D-32
wfDynamicUserAssign, D-33
FYI assignee task participant, 15-35
group rules integration definition, 15-3
group vote task participant, 15-27
group voting details, 15-29
Human Task editor definition, 15-3, 15-12, 15-13
identification key, 15-59
identity service, 15-6
in Oracle JDeveloper, 2-19
including the task history of other tasks, 15-59, 15-64
inviting additional task participants, 15-26, 15-32, 15-34
Java support, 15-97
management chain task participant, 15-30
multilingual settings, 15-48, 15-83
never expire policy, 15-41
notification definition, 15-2
notification contents, 15-82
notification preferences, 15-43
notification service, 15-4, 15-6, 15-81
notifications, 15-80
notifying recipients of changes to task status, 15-44
number of task approvers, 15-32
opening a definition already associated with a BPEL process, 15-54
Oracle BPEL Worklist Application, 15-4
outcome-based modeling, 15-64
overriding default exception management, 15-51
overriding default system actions, 15-49
overview, 15-97
overview of modeling process, 15-12
participant types, 15-11
participant types of a workflow, 15-9
procurement process example, 15-5
proxy support, 15-10
renew after policy, 15-42
reports definition, 15-4
routing slip definition, 15-3, 15-26
runtime config service, 15-6
samples, 1-8
scope name and global task variable name, 15-58
securing notifications, 15-46, 15-85
security model, 15-98
sequential list of approvers task participant, 15-32
setting up reminders, 15-46
sharing attachments and comments with task participants, 15-29, 15-36
simple task display form, 15-68
simple task display form payload, 15-68
single approver task participant, 15-24
SOAP support, 15-97
specifying a task initiator and task priority, 15-35
specifying a task title, 15-55
specifying callback classes, 15-51
specifying class names, 15-37
specifying task approvers, 15-23
specifying task parameters, 15-56
storage of task history for auditing, 15-3
style sheets in attachments, 15-48
support for group rules, 15-3
support for identity service, 15-4, 15-102
support for JSP-based forms, 15-3
support for multiple participant types, 15-4
support for task assignment and routing, 15-3
support for task delegation, escalation, and reapproval, 15-3
support for task expiration and automatic renewal, 15-3
support for user rules, 15-3
task definition, 15-2
task assignment and routing definition, 15-3
task attachments with e-mail notifications, 15-46, 15-84

Index-29
Index-30

Oracle BPEL Worklist Application request status, 16-11
Oracle BPEL Worklist Application requests for more information, 16-15
Oracle BPEL Worklist Application routing, 16-15
Oracle BPEL Worklist Application Task Actions list, 16-10
Oracle BPEL Worklist Application task reassignment, 16-15
Oracle BPEL Worklist Application user and group information, 16-39
parallel tasks, 16-17
payload viewing, 16-13
request status, 16-11
requesting more information, 16-15
routing, 16-15
task actions, 16-3, 16-10
task reassignment, 16-15
use cases, 16-1
user and group information, 16-39
writeBinaryToFile function
description, D-16
WS-Addressing, 5-7
definition, 5-7
sending correlation IDs, 5-6
WSDL cache
clearing, 19-14
WSDL files
definition, 2-6
differences between document-literal styles and RPC styles, 3-2, 3-22
location for identity service, 15-98
location for notification service, 15-98
location for runtime config service, 15-98
location for task metadata service, 15-97
location for task query service, 15-97
location for task service, 15-97
location for user metadata service, 15-98
modifying to generate a fault, 8-5
specifying when creating a partner link, 2-17
WSIF bindings
decision service, 18-47
definition, 9-1
for using Java code in a BPEL process, 9-1
Java binding service using XML DOM elements, 9-3
Java binding service using XML facades, 9-2
Java binding service using XML simple types, 9-2
Java throwing a WSDL fault from a Java binding, 9-4
WSIF providers
adding a custom WSIF provider, 3-23
WSIL directories
browsing for remote services, 4-4
browsing for services, 4-4

X

XML assert
overview, 20-3
XML data in BPEL, 3-2
XML data manipulation
bpelx:append extension, 3-10
bipelx:copyList extension, 3-15
bipelx:insertAfter extension, 3-12
bipelx:insertBefore extension, 3-11
bipelx:remove extension, 3-12
bipelx:rename extension, 3-13
bipelx:validate extension, 3-16
XML documents
manipulating, 3-2, 3-4
overview, 3-2, 3-4
XML facets
definition, 9-7
generating with schemac, 9-7, 19-29
Java binding service using, 9-2
Java embedding, 9-7
tutorials, 9-7
XML schemas
in BPEL tests, 20-32
message types and variable types, 3-1
XML variables
tutorials, 1-10
XPath, 3-2
 XPath expressions
assigning numeric values, 3-6
Boolean expressions in switch activities, 7-4
dynamically creating another XPath expression, 3-17
dynamically setting e-mail addresses and telephone numbers, 14-13
editing in transformations, 13-11
tutorials, 13-11
examples, 3-3
fetching a data sequence element, 3-17
in conditional branching logic, 7-1
specifying a task owner, 15-20
specifying task approvers, 15-23
XPath functions
creating a custom XPath function, D-45
examples, 3-4
in transformations, 13-8
indexing methods, 3-17
mathematical calculations, 3-7
references, 1-10
using custom functions within assign activities, 1-9
XPath queries
copying data, 3-6
tutorials, 3-3
examples, 3-3
references, 1-10
XQuery, 3-2, 3-4
XQuery transformations
tutorials, 1-11
XSL
custom task display form, 15-77
xml choose
conditional processing, 13-13
xml if
conditional processing, 13-12
XSL map
creating from an XSL stylesheet, 13-1
XSL stylesheet
creating an XSL map, 13-1
XSLMapper
demos, 1-9
XSLT, 3-2, 3-4
XSLT constructs
adding in transformations, 13-12
XSLT Mapper
adding XSLT constructs, 13-12
auto mapping, 13-15
auto mapping with confirmation, 13-17
chaining functions, 13-10
correcting memory errors when generating reports, 13-27
creating a map file, 13-5
creating a map file from imported schemas, 13-3
creating a new map file, 13-2
creating a transform activity, 13-1
creating an XSL map from an XSL stylesheet, 13-1
customizing sample XML generation for transformations, 13-27
demos, 1-9
dictionaries, 13-19
demos, 1-9
dictionaries, 13-19
demographics, 13-13
directories, 13-13
directories, 13-13
directories, 13-13
directories, 13-13
directories, 13-13
directories, 13-13
directories, 13-13
directories, 13-13
directories, 13-13
directories, 13-13
directories, 13-13
year-from-dateTime function
description, D-22
