Oracle® BPEL Process Manager

Developer's Guide

10*g* Release 2 (10.1.2)

Part No. B14448-01

October 24, 2005

ORACLE®

Oracle BPEL Process Manager Developer’s Guide, 10*g* Release 2 (10.1.2)

Part No. B14448-01

Primary Author: Deanna Bradshaw, Mark Kennedy, Craig West

Contributor: Oracle BPEL Process Manager development, product management, and quality assurance teams

Contents

Send Us Your Comments xxi

Preface xxiii

- Intended Audience xxiv
- Documentation Accessibility xxiv
- Structure xxiv
- Related Documents xxvii
- Conventions xxviii

Part I Introduction and Concepts

1 Introduction to Oracle BPEL Process Manager

- **What Is BPEL?** 1-2
- **What Is Oracle BPEL Process Manager?** 1-2
- **What Is the BPEL Designer?** 1-3
  - JDeveloper BPEL Designer 1-3
  - Eclipse BPEL Designer 1-4
- **How to Use This Guide** 1-5
- **Tutorials and Demonstrations** 1-7
- **Summary** 1-12

2 Getting Started with Oracle BPEL Process Manager

- **Overview of Oracle BPEL Process Manager Components** 2-2
- **Starting Oracle BPEL Process Manager Components** 2-2
- **Overview of BPEL Designer Environments** 2-3
  - Overview of JDeveloper BPEL Designer 2-3
    - Applications Navigator 2-5
    - Diagram View Window 2-6
    - Source Window 2-8
    - Component Palette 2-9
    - Property Inspector 2-10
    - Structure Window 2-11
    - Log Window 2-12
  - Overview of Eclipse BPEL Designer 2-13
    - Navigator 2-15

Process Map and Overview Windows 2-15
BPEL Source Window 2-17
BPEL Inspector 2-17
Log Window 2-17
BPEL Palette 2-18
Overview of Activities 2-19
Overview of Partner Links 2-20
Overview of Oracle BPEL Server 2-21
Overview of Oracle BPEL Console 2-21
Overview of Oracle BPEL Process Manager Services 2-22
Overview of Oracle BPEL Process Manager Technology Adapters 2-23
Summary 2-25

3 Building a Simple BPEL Process

Overview of Building a Simple BPEL Process 3-1
Creating a New BPEL Project Using Eclipse BPEL Designer 3-2
Browsing a New Project 3-4
Viewing the WSDL Interface of a BPEL Process 3-5
Switching Between the Overview, Process Map, and Source Code 3-6
Viewing an Overview of a BPEL Process 3-6
Viewing a Detailed Process Map 3-7
Viewing BPEL Source Code 3-8
Reviewing the BPEL Source Code 3-9
Understanding the Sequence Activity 3-11
Understanding the Assign Activity 3-11
Adding an Assign Activity to the Process Map 3-12
Step 1: Viewing the Process Map 3-12
Step 2: Inserting an Assign Activity 3-12
Step 3: Adding a Copy Rule 3-14
Step 4: Defining the From Part (Source) of the Copy Rule 3-15
Step 5: Defining the To Part (Destination) of the Copy Rule 3-16
Compiling and Deploying the BPEL Process 3-16
Testing the BPEL Process 3-17
Summary 3-19

Part II Reviewing Key BPEL Development Concepts and Code Samples

4 Manipulating XML Data in BPEL

How XML Data Works in BPEL 4-2
About Data Manipulation and XPath Standards 4-2
Initializing a Variable with Expression Constants or Literal XML 4-4
Copying Between Variables 4-5
Accessing Fields within Complex Type Variables 4-6
Assigning Numeric Values 4-6
Mathematical Calculations with XPath Standards 4-7
Assigning String Literals 4-7

Concatenating Strings 4-8
Assigning Boolean Values 4-8
Assigning Date or Time 4-9
Manipulating Attributes 4-9
Manipulating XML Data Sequences/Arrays 4-10
- Statically Indexing into an XML Data Sequence 4-11
- Determining Sequence Size 4-12
- Dynamically Indexing by Applying a Trailing XPath to an Expression 4-12
  - Dynamic Indexing Example 4-12
  - Appending New Items to a Sequence 4-13
  - Merging Data Sequences 4-14
  - Dynamically Indexing with the BPEL getElement Function 4-14
  - Merging Data Sequences/Arrays 4-15
  - Appending New Items to a Sequence/Array 4-15

Converting from a String to an XML Element 4-16
Differences Between Document-Style and RPC-Style WSDL Files 4-16
Summary 4-17

5 Invoking a Synchronous Web Service

Use Case 5-2
Synchronous Service Concepts 5-2
- Examples 5-2
- The Partner Link 5-3
- Port Types 5-4
- partnerLinkTypes for Synchronous Services 5-4
- UDDI and WSIL Directories 5-5
- The Invoke Activity 5-5

Calling a Synchronous Service 5-6
Summary 5-7

6 Calling an Asynchronous Web Service

Introduction 6-2
Use Case 6-3
Understanding Asynchronous Callback Concepts 6-3
- partnerLinkTypes for Asynchronous Services 6-4
- Calling the Service from BPEL 6-5
- Invoke and Receive Activities 6-5
- Correlations 6-6
  - WS-Addressing 6-7
    - TCP Tunneling 6-8
  - Correlation Sets 6-8
- The Reply Activity 6-8
- Dehydration 6-9

Calling an Asynchronous Service 6-9
- Adding a Partner Link for an Asynchronous Service 6-9
- Adding an Invoke Activity 6-10

Adding a Receive Activity ... 6-12
Performing Additional Activities ... 6-13
Questions and Answers ... 6-13
Summary ... 6-13

7 Parallel Flow

Introduction ... 7-2
Use Case ... 7-2
Understanding Parallel Flow Concepts ... 7-2
Defining a Parallel Flow ... 7-3
The flowN Activity ... 7-4
BPEL Code Example of the FlowN Activity ... 7-7
Summary ... 7-8

8 Conditional Branching

Introduction ... 8-2
Use Case ... 8-2
Understanding Conditional Branching Concepts ... 8-2
Conditional Branching ... 8-2
Adding a Switch Activity ... 8-3
The While Activity ... 8-4
Summary ... 8-5

9 Fault Handling

Introduction ... 9-2
Use Case ... 9-2
Defining a Fault Handler ... 9-2
Taxonomy of BPEL Faults ... 9-4
Using the Scope Activity ... 9-4
Throwing Internal Faults ... 9-7
Returning External Faults ... 9-7
Returning a Fault in a Synchronous Interaction ... 9-8
Returning a Fault in an Asynchronous Interaction ... 9-8
Fault Handler Within a Scope ... 9-8
The Empty Activity at runtime ... 9-9
Compensation ... 9-9
The Terminate Activity ... 9-10
Catching Run-Time Faults Example ... 9-11
Eclipse BPEL Designer Example ... 9-11
Summary ... 9-15

10 Incorporating Java/J2EE Code in BPEL Processes

Introduction ... 10-2
Use Case ... 10-2
Using Java Code with WSIF Binding ... 10-2
Using Java Code Wrapped as a SOAP Service ... 10-2

Embedding Java Code in BPEL 10-3
The bpelx:exec Tag 10-3
XML Facade 10-3
bpelx:exec Built-in Methods 10-4
JDeveloper BPEL Designer Example 10-5
Summary 10-6

11 Events and Timeouts

Introduction 11-2
Use Case 11-2
The pick Activity 11-2
The Wait Activity 11-4
JDeveloper BPEL Designer Example 11-4
Synchronous Processes 11-7
Summary 11-7

12 Invoking a BPEL Process

Introduction 12-2
Use Case 12-2
Sending Messages to a BPEL Process from a Java/JSP Application 12-2
Invoking a BPEL Process with the Generic Java API 12-3
Connecting to Oracle BPEL Process Manager with the Locator Class 12-3
Passing XML Messages through Java 12-3
Invoking a Two-Way Operation through the Java API 12-4
Invoking a One-Way Operation through Java API 12-5
Retrieving Status/Results from Asynchronous BPEL Processes 12-5
Using the Java API from a Remote Client 12-6
Invoking a BPEL Process with the Web Service/SOAP Interface 12-6
Summary 12-7

13 Interaction Patterns

Introduction 13-2
One-Way Message 13-2
BPEL Process as the Client 13-2
BPEL Process as the Service 13-2
Synchronous Interaction 13-2
BPEL Process as the Client 13-3
BPEL Process as the Service 13-3
Asynchronous Interaction 13-3
BPEL Process as the Client 13-4
BPEL Process as the Service 13-4
Asynchronous Interaction with Timeout 13-4
BPEL Process as the Client 13-5
BPEL Process as the Service 13-5
Asynchronous Interaction with a Notification Timer 13-5
BPEL Process as the Client 13-6

BPEL Process as the Service 13-6
One Request, Multiple Responses 13-6
BPEL Process as the Client 13-7
BPEL Process as the Service 13-7
One Request, One of Two Possible Responses 13-7
BPEL Process as the Client 13-8
BPEL Process as the Service 13-8
One Request, a Mandatory Response, and an Optional Response 13-8
BPEL Process as the Client 13-9
BPEL Process as the Service 13-9
Partial Processing 13-9
BPEL Process as the Client 13-10
BPEL Process as the Service 13-10
Third-Party Interactions 13-10
Summary 13-11

Part III Oracle BPEL Process Manager Services

14 XSLT Mapper and Transformations

Use Case for Transformation 14-2
Creating a Transform Activity 14-2
The XSLT Mapper 14-3
Notes on the Mapper 14-5
Step 1: Creating an XSL Map 14-5
Step 2: Using the Mapper 14-5
Simple Copy by Linking Nodes 14-6
Setting Constant Values 14-6
Functions 14-7
Editing Function Parameters 14-8
Chaining Functions 14-9
Named Templates and User-Defined Functions 14-9
Editing XPath Expressions 14-9
Adding XSLT Constructs 14-11
Conditional Processing with xsl:if 14-11
Conditional Processing with xsl:choose 14-12
Handling Repetition or Arrays 14-13
Auto Mapping 14-14
Auto Map with Confirmation 14-16
Generating Dictionaries 14-18
Step 3: Testing the Map 14-18
Test Window 14-19
Generating Reports 14-20
Summary 14-21

15 Oracle BPEL Process Manager Notification Service

Use Cases for Notification Service 15-1

Overview of Notification Service Concepts 15-1
Configuring Notification Service in JDeveloper BPEL Designer 15-2
The E-mail Notification Channel 15-4
Setting E-mail Attachments 15-6
Configuring the E-mail Server 15-8
Example ns_emails.xml File 15-9
The Voice Notification Channel 15-10
Configuring the Wireless Service Provider for Voice 15-11
Example ns_iaswconfig.xml File 15-11
The SMS Notification Channel 15-12
Configuring the Wireless Service Provider for SMS 15-13
Setting E-mail Addresses and Telephone Numbers Dynamically 15-13
Selecting Notification Recipients by Browsing the User Directory 15-13
Starting Business Processes with the E-mail Activation Agent 15-14
Summary 15-15

16 Oracle BPEL Process Manager Workflow Services

Overview of Workflow Services 16-1
Workflow Functionality: A Procurement Process Example 16-3
Workflow Services Components 16-4
Use Cases for Workflow Services 16-6
Assigning a Task to a User or Role 16-6
Using the Various Flow Patterns 16-7
Escalation, Expiration, and Delegation 16-7
The Worklist Application 16-8
Workflow Patterns 16-8
The Modeling Process 16-10
Editing or Deleting a Workflow 16-11
Task Details and Configurations 16-11
Task Attributes 16-11
Task Outcomes 16-13
Advanced Task Configurations 16-14
Flex Fields 16-15
Restricted Actions 16-16
Version-Tracking Attributes 16-17
Task Notifications and Reminders 16-19
Resource Bundles 16-19
Task Assignment 16-19
Task Assignment Evaluation 16-20
Task Assignment Based on External Services 16-21
Assigning a Task to a Specific User of a Role and Marking It As Acquired 16-21
Setting Task Assignees from a Dynamic Delimited String 16-22
Selecting Users or Groups by Browsing the User Directory 16-23
Adding a Task Attachment from a Business Process 16-26
Actions Performed on a Task 16-29
Simple Workflow 16-31
Use Case 16-32

Customizations for Simple Workflow 16-32
Simple Workflow with Automatic Escalation 16-33
Use Case 16-35
Pattern-Specific Parameters 16-35
Customizations for Simple Workflow with Automatic Escalation 16-36
Simple Workflow with Automatic Renewal 16-37
Use Case 16-39
Pattern-Specific Parameters 16-39
Customizations for Simple Workflow with Automatic Renewal 16-39
Sequential Workflow 16-40
Use Cases 16-43
Pattern-Specific Parameters 16-43
Customizations for Sequential Workflow 16-45
Sequential Workflow with Escalation 16-46
Use Case 16-47
Pattern-Specific Parameters 16-48
Customizations for Sequential Workflow with Escalation 16-48
Parallel Workflow 16-49
Use Case 16-51
Pattern-Specific Parameters 16-51
Customizations for Parallel Workflow 16-52
Parallel Workflow with Final Reviewer 16-52
Use Case 16-53
Pattern-Specific Parameters 16-53
Customizations for Parallel Workflow with Final Reviewer 16-53
Adhoc Workflow 16-54
Use Case 16-55
Customizations for Adhoc Workflow 16-55
FYI Tasks 16-55
Use Case 16-55
Customization for FYI Tasks 16-56
The User Task 2.0 Macro 16-56
Task Continuations 16-56
Use Case 16-58
Pattern-Specific Parameters 16-58
Customization for Task Continuations 16-58
Outcome-Based Modeling 16-59
Payload Updates 16-59
Case Statements for Other Task Conclusions 16-59
Task Notifications 16-60
Channels Used for Notification 16-61
Notification Messages 16-61
E-mail Approval 16-62
Reminders 16-63
Payload Display 16-64
Autogenerated JSP 16-65
Customizing the Autogenerated JSP 16-67

Customizing the Mapping File 16-67
Customizing the Default JSP 16-69
Multibyte Payload in the Task Detail JSP 16-71
Deploying the Autogenerated JSP 16-71
XSL 16-71
The Custom JSP URL 16-72
APIs 16-73
Customizing the Complete Task JSP 16-73
Configuration for Task Service 16-73
Autorelease Duration 16-73
Actionable E-mail Accounts 16-74
Worklist Application URL 16-74
Identity Service 16-75
Identity Service Providers 16-76
The JAZN Provider 16-76
XML-Based JAZN Provider Type 16-77
LDAP-Based JAZN Provider Type (Oracle Internet Directory) 16-77
Third-Party LDAP Server 16-77
Custom User Repository Plug-ins 16-77
Creating Users and Groups 16-77
User and Role Properties 16-78
Configuring Identity Service 16-79
Structure of the Identity Service Configuration File 16-79
connection Element 16-80
userControls and roleControls Elements 16-81
provider Element 16-83
Configuration for the XML-Based JAZN Provider 16-83
Configuration for the LDAP-Based JAZN Provider (OID) 16-83
OID Configuration 16-84
Middle-Tier Configuration 16-85
Configuration for a Third-Party LDAP Server 16-86
Configuration for CUSTOM User Repository Plug-ins 16-89
Workflow-Related XPath Extension Functions 16-89
Approver Functions 16-95
Approver Function Syntax 16-95
Approver Function Examples 16-96
Vacation Request Example 16-98
Prerequisites 16-98
Getting Started: Modeling the Vacation Request Process 16-99
Running the Example 16-104
Summary 16-105

17 Worklist Application

Use Cases for the Worklist Application 17-1
Overview of Worklist Application Concepts 17-2
Logging In to the Worklist Application 17-4
Features of the Sample Worklist Application 17-4

Task Actions 17-10
Routing 17-12
Requesting More Information 17-13
Task History and Sequence (Version) Numbers 17-14
The Payload 17-17
Comments 17-19
Attachments 17-20
Reassignment 17-22
Parallel Tasks 17-23
Flex Fields and Task Fields Updates 17-24
Request Status 17-26
Error Information 17-28
User and Group Information 17-29
Advanced Search 17-31
Determining Action Permissions 17-32
How Changes to a Workflow Appear in the Worklist Application 17-33
Accessing the Worklist Application in Local Languages 17-33
Customizing the Worklist Application 17-34
Worklist Application Architecture 17-34
Login Page 17-35
Header Info 17-36
Task Home (Listing) Page 17-37
Task Details Page 17-37
Additional Pages 17-38
Configuration Parameters 17-38
Controlling Access to Information and Actions for Different Users 17-39
Building a Worklist Application Using the Worklist Service APIs 17-40
Worklist Service APIs 17-41
Example: Reassigning a Task in a Worklist Application 17-45
Building a Worklist Application Using the Worklist Service Remote APIs 17-45
Summary 17-46

18 Sensors

Use Cases for Sensors 18-1
Overview of Sensor Concepts 18-1
Sensor Public Views 18-2
BPEL Reporting Schema 18-2
Implementing Sensors and Sensor Actions in JDeveloper BPEL Designer 18-9
Configuring Sensors 18-10
Configuring Sensor Actions 18-12
Creating a Custom Data Publisher 18-15
Registering the Sensors and Sensor Actions in bpel.xml 18-19
Sensors and Oracle BPEL Console 18-19
Viewing Sensor and Sensor Action Definitions 18-19
Viewing Sensor Data 18-20
Summary 18-21

Part IV Development Life Cycle

19 BPEL Process Deployment and Domain Management

Compiling and Deploying a BPEL Process 19-2
Compiling and Deploying on JDeveloper BPEL Designer 19-2
Compiling Without Deploying on JDeveloper BPEL Designer 19-4
Compiling and Deploying on Eclipse BPEL Designer 19-5
BPEL Suitcase JAR File 19-6
Creating and Managing a BPEL Domain 19-8
Changing the Default Domain Password 19-8
Changing Oracle BPEL Admin Console Password 19-8
Creating a BPEL Domain 19-9
Changing Oracle BPEL Server Mode 19-9
Deploying a BPEL Suitcase to a Specific Domain 19-10
Location of BPEL JAR Suitcase Files in a Specific Domain 19-11
Undeploying a BPEL Process from a Specific Domain 19-11
Viewing BPEL Processes in Oracle BPEL Console 19-12
Dashboard Tab: Viewing Processes 19-12
BPEL Processes Tab: Managing the Process Life Cycle 19-13
Process Life Cycle Recommendations for a Development Environment 19-15
Process Life Cycle Recommendations for a Production Environment 19-15
Example: Life Cycle of Processes 19-16
Instances Tab: Viewing Process Instances 19-21
Activities Tab: Viewing Process Activities 19-21
Build and Command Line Tools 19-23
Apache Ant 19-23
bpelc 19-23
Examples 19-24
schemac 19-24
Examples 19-25
Summary 19-26

Part V Reference Information

A Troubleshooting and Workarounds

Troubleshooting Sensors—The Custom Data Publisher A-1
Poor JMS Performance When Creating or Destroying Connections A-1
Data Publisher Is Not Working A-1
Data Publisher Works, But Business Process Runs Slowly A-2
Caching Data in the Data Publisher Is Not Supported A-2
Unexpected Errors in the Data Publisher A-2
Data Extracted to XML Is Difficult to Work With A-2
Troubleshooting Oracle BPEL Worklist Application A-3
Not Able to Log In to the Worklist Application A-3
Information Is Displayed in a Different Language A-3
Dates and Times Are Displayed Incorrectly A-3

The User Is Not Permitted to Perform an Action A-3
Expected Task Is Not Listed Under Task Titles A-3
Summary A-4

B Workflow and Notification Reference

Task Manager Service WSDL Operations B-1
Task Routing Service WSDL Operations B-2
Notification WSDL Operations B-2
Identify Service Operations B-3
Task Action Handler Business Process B-3
Summary B-3

C JDeveloper BPEL Designer Activities

Validating when Loading a Process Diagram C-2
Activities Overview C-2
Tabs Common to Many Activities C-3
Assign Activity C-3
Catch Activity C-4
Compensate Activity C-5
Empty Activity C-6
Flow Activity C-6
FlowN Activity C-7
Invoke Activity C-8
Java Embedding Activity C-9
Notification Activity C-10
PartnerLink Activity C-10
Pick Activity C-11
Receive Activity C-12
Reply Activity C-13
Sequence Activity C-14
Scope Activity C-15
Switch Activity C-15
Terminate Activity C-16
Throw Activity C-17
Transform Activity C-17
User Task C-18
Wait Activity C-19
While Activity C-20
Summary C-21

D User Task 2.0 Macro

Introduction to User Task 2.0 Macro D-2
BPEL User Task Use Case D-2
The TaskManager Service D-2
Integrating the TaskManager Service into a BPEL Process D-6
Defining a Partner Link for the TaskManager Service D-6

Declare and Initialize the Task Document D-7
Initiate the Task D-7
Task Completion D-8
Using Eclipse BPEL Designer to Integrate the TaskManager Service D-8
Creating a User Interface for the Task D-12
List the Assigned Tasks D-12
Display the Payload Data for a Task D-14
Update the Payload Data and Complete the Task D-15
Additional Capabilities of the TaskManager Service D-16
Enabling Expiration/Timeouts for Tasks D-16
Sending Notifications D-17
Reassigning Tasks D-18
Assigning Tasks to Groups and Resolving Roles D-18
Summary D-18

E Deployment Descriptor Properties

Deployment Descriptor Preference Properties E-1
Defining a Preference Property E-1
Updating a Preference at Run Time E-2
Getting the Value of a Preference within a BPEL Process E-3
Encrypting a Preference Value E-3
Use Case for Preference Properties E-4
Deployment Descriptor Configuration Properties E-4
Defining a Configuration Property E-4
Summary E-8

F Demo User Community

Setting Up JAZN Demo Users F-1
Demo Users and Roles F-1
Using the Demo User Community in the Order Booking Tutorial F-3
Summary F-4

G XPath Extension Functions

XPath Extension Functions Available to BPEL Processes G-1
abs G-1
Property IDs G-1
add-dayTimeDuration-to-dateTime G-1
Property IDs G-2
addChildNode G-2
Property IDs G-2
addQuotes G-2
Property IDs G-2
appendToList G-3
Property IDs G-3
authenticate G-3
Property IDs G-3

batchProcessActive G-3
Property IDs G-4
batchProcessCompleted G-4
Property IDs G-4
clearTaskAssignees G-4
Property IDs G-4
compare G-4
Property IDs G-5
compare-ignore-case G-5
Property IDs G-5
copyList G-5
Property IDs G-6
countNodes G-6
Property IDs G-6
create-delimited-string G-6
Property IDs G-6
create-nodeset-from-deliminated-string G-7
Property IDs G-7
createDeliminatedString G-7
Property IDs G-7
current-date G-7
Property IDs G-8
current-dateTime G-8
Property IDs G-8
current-time G-8
Property IDs G-8
day-from-dateTime G-8
Property IDs G-9
doc G-9
Property IDs G-9
ends-with G-9
Property IDs G-9
format G-9
Property IDs G-10
format-dateTime G-10
Property IDs G-10
format-string G-10
Property IDs G-10
formatDate G-10
Property IDs G-11
genEmptyElem G-11
Property IDs G-11
generate-guid G-11
Property IDs G-11
generateGUID G-11
Property IDs G-12
get-content-as-string G-12

Property IDs G-12
get-localized-string G-12
Property IDs G-13
getChildElement G-13
Property IDs G-13
getContentAsString G-13
Property IDs G-13
getConversationId G-13
Property IDs G-13
getCreator G-14
Property IDs G-14
getCurrentDate G-14
Property IDs G-14
getCurrentDateTime G-14
Property IDs G-14
getCurrentTime G-14
Property IDs G-15
getDomainId G-15
Property IDs G-15
getElement G-15
Property IDs G-15
getGroupIdsFromGroupAlias G-15
Property IDs G-15
getGroupProperty G-16
Property IDs G-16
getInstanceId G-16
Property IDs G-16
getLinkStatus G-16
Property IDs G-16
getManager G-17
Property IDs G-17
getMessage G-17
Property IDs G-17
getNodeValue G-17
Property IDs G-17
getNodes G-18
Property IDs G-18
getNumberOfTaskApprovals G-18
Property IDs G-18
getPreference G-18
Property IDs G-18
getPreviousTaskApprover G-18
Property IDs G-19
getProcessId G-19
Property IDs G-19
getProcessOwnerId G-19
Property IDs G-19

getProcessURL G-19
    Property IDs G-19
getProcessVersion G-19
    Property IDs G-20
getReportees G-20
    Property IDs G-20
getTaskAttachmentByIndex G-20
    Property IDs G-20
getTaskAttachmentByName G-20
    Property IDs G-20
getTaskAttachmentContents G-21
    Property IDs G-21
getTaskAttachmentsCount G-21
    Property IDs G-21
getTaskAutoReleaseDuration G-21
    Property IDs G-21
getTaskReminderDuration G-21
    Property IDs G-22
getUserAliasId G-22
    Property IDs G-22
getUserIdsFromGroupAlias G-22
    Property IDs G-22
getUserProperty G-22
    Property IDs G-23
getUserRoles G-23
    Property IDs G-23
getUsersInGroup G-23
    Property IDs G-24
getVariableData G-24
    Property IDs G-24
getVariableProperty G-24
    Property IDs G-25
hours-from-dateTime G-25
    Property IDs G-25
implicit-timezone G-25
    Property IDs G-25
index-within-string G-25
    Property IDs G-26
integer G-26
    Property IDs G-26
isUserInRole G-26
    Property IDs G-26
last-index-within-string G-26
    Property IDs G-27
lookupUser G-27
    Property IDs G-27
left-trim G-27

Property IDs G-27
listUsers G-28
Property IDs G-28
lookup-table G-28
Property IDs G-28
lookup-xml G-29
Property IDs G-29
lookupUser G-29
Property IDs G-29
lookupGroup G-29
Property IDs G-30
lower-case G-30
Property IDs G-30
matches G-30
Property IDs G-30
max-value-among-nodeset G-30
Property IDs G-31
mergeChildNodes G-31
Property IDs G-31
min-value-among-nodeset G-31
Property IDs G-32
minutes-from-dateTime G-32
Property IDs G-32
month-from-dateTime G-32
Property IDs G-32
parseEscapedXML G-32
Property IDs G-32
processXSLT G-33
Property IDs G-33
processXSLT G-33
Property IDs G-33
processXSQL G-33
Property IDs G-34
query-database G-34
Property IDs G-34
readFile G-34
Property IDs G-34
right-trim G-34
Property IDs G-35
search G-35
Property IDs G-35
seconds-from-dateTime G-35
Property IDs G-35
sequence-next-val G-35
Property IDs G-36
setNodeValue G-36
Property IDs G-36

square-root G-36
Property IDs G-37
subtract-dayTimeDuration-from-dateTime G-37
Property IDs G-37
timezone-from-dateTime G-37
Property IDs G-37
translateFromNative G-38
Property IDs G-38
translateToNative G-38
Property IDs G-38
upper-case G-38
Property IDs G-38
year-from-dateTime G-39
Property IDs G-39
Summary G-39

Index

Send Us Your Comments

Oracle BPEL Process Manager Developer's Guide, 10*g* Release 2 (10.1.2)

Part No. B14448-01

Oracle welcomes your comments and suggestions on the quality and usefulness of this publication. Your input is an important part of the information used for revision.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where?
- Are the examples correct? Do you need more examples?
- What features did you like most about this manual?

If you find any errors or have any other suggestions for improvement, please indicate the title and part number of the documentation and the chapter, section, and page number (if available). You can send comments to us in the following ways:

- Electronic mail: appserverdocs_us@oracle.com
- FAX: 650-506-7375. Attn: Oracle Application Server Documentation Manager
- Postal service:

  Oracle Corporation
  Oracle Application Server Documentation
  500 Oracle Parkway, Mailstop 1op4
  Redwood Shores, CA 94065
  USA

If you would like a reply, please give your name, address, telephone number, and electronic mail address (optional).

If you have problems with the software, please contact your local Oracle Support Services.

Preface

This manual describes how to use Oracle BPEL Process Manager.

This preface contains the following topics:

- Intended Audience
- Documentation Accessibility
- Structure
- Related Documents
- Conventions

Intended 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 should 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.

Structure

This guide consists of the following chapters and appendices:

Part I, "Introduction and Concepts"

This part introduces Oracle BPEL Process Manager.

Chapter 1, "Introduction to Oracle BPEL Process Manager"

This chapter provides a brief introduction to the Business Process Execution Language (BPEL), how Oracle BPEL Process Manager supports BPEL, and the types of BPEL designers available with Oracle BPEL Process Manager that enable you to design BPEL processes (either JDeveloper BPEL Designer and Eclipse BPEL Designer). An overview of how to use the information in this guide and references to additional tutorials and demonstrations installed with Oracle BPEL Process Manager is also provided.

Chapter 2, "Getting Started with Oracle BPEL Process Manager"

This chapter describes how to get started with key Oracle BPEL Process Manager components, including the JDeveloper BPEL Designer, Eclipse BPEL Designer, and Oracle BPEL Console. Key design concepts are also described.

Chapter 3, "Building a Simple BPEL Process"

This chapter discusses how to install the JDeveloper BPEL Designer and how to build, deploy, and test a simple BPEL process, a synchronous Hello World application.

Part II, "Reviewing Key BPEL Development Concepts and Code Samples"

This part introduces key BPEL development concepts and code samples.

Chapter 4, "Manipulating XML Data in BPEL"

This chapter provides details about the role that XPath functionality plays in manipulating XML data.

Chapter 5, "Invoking a Synchronous Web Service"

This chapter discusses the components necessary to perform a synchronous callback, examines how these components are coded, and shows how to set up a synchronous callback using Eclipse BPEL Designer.

Chapter 6, "Calling an Asynchronous Web Service"

This chapter describes the components necessary to perform asynchronous messaging.

Chapter 7, "Parallel Flow"

This chapter describes how to use parallel flows to enable BPEL to perform multiple tasks at the same time.

Chapter 8, "Conditional Branching"

This chapter describes how to use conditional branching decision points to control the flow of execution of the process.

Chapter 9, "Fault Handling"

This chapter describes how to use fault handling to enable a BPEL process to manage error messages or other exceptions returned by services.

Chapter 10, "Incorporating Java/J2EE Code in BPEL Processes"

This chapter demonstrates how a developer can embed a section of Java code into a BPEL process.

Chapter 11, "Events and Timeouts"

This chapter describes how to enable a BPEL process to time out or give up waiting and continue with the rest of a flow after a certain amount of time.

Chapter 12, "Invoking a BPEL Process"

This chapter describes how a Java/JSP application can call a BPEL process to perform functions or use services.

Chapter 13, "Interaction Patterns"

This chapter identifies common interaction patterns between a BPEL process and another application, and shows the best use practices for each.

Part III, "Oracle BPEL Process Manager Services"

This part describes how Oracle BPEL Process Manager extends key BPEL development concepts to include support for services.

Chapter 14, "XSLT Mapper and Transformations"

This chapter lists important features of the XSLT mapper and provides step-by-step instructions for mapping a sample purchase order schema to an invoice schema.

Chapter 15, "Oracle BPEL Process Manager Notification Service"

This chapter describes how to use the notification service.

Chapter 16, "Oracle BPEL Process Manager Workflow Services"

This chapter defines some basic workflow services terms. Workflow services features are introduced and then discussed in more detail. The discussion of how to model a workflow is followed by a vacation request workflow example.

Chapter 17, "Worklist Application"

This chapter describes how to use the Oracle BPEL Worklist Application. You can take actions on tasks such as approving an employee vacation request, evaluating a job applicant, or escalating a purchasing decision. Based on your user profile, you see all the tasks relevant to you and can specify search criteria for displaying tasks.

Chapter 18, "Sensors"

This chapter describes how to use and set up sensors for a BPEL process.

Part IV, "Development Life Cycle"

This part describes how to run BPEL processes from the Oracle BPEL Console.

Chapter 19, "BPEL Process Deployment and Domain Management"

This chapter discusses how to deploy processes.

Part V, "Reference Information"

This part provides reference details about activities, transformation functions, and troubleshooting issues

Appendix A, "Troubleshooting and Workarounds"

The appendix describes Oracle BPEL Process Manager troubleshooting methods.

Appendix B, "Workflow and Notification Reference"

This appendix describes workflow and notification service operations.

Appendix C, "JDeveloper BPEL Designer Activities"

This appendix describes the activities available for use when designing a BPEL process in JDeveloper BPEL Designer.

Appendix D, "User Task 2.0 Macro"

This appendix describes the User Tasks 2.0 macro, which supports user tasks from release 2.0. The user task 2.0 macro is available for backward compatibility and is replaced in this release.

Appendix E, "Deployment Descriptor Properties"

This chapter discusses deployment descriptor preference properties and deployment descriptor configuration properties.

Appendix F, "Demo User Community"

This appendix describes the demo user community for Oracle BPEL Process Manager.

Appendix G, "XPath Extension Functions"

This appendix describes the XPath extension functions.

Related Documents

For more information, see the following Oracle resources:

- *Oracle Adapters for Files, FTP, Databases, and Enterprise Messaging User's Guide*
- *Oracle BPEL Process Manager Quick Start Guide*
- *Oracle BPEL Process Manager Order Booking Tutorial*

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:

```
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbizspec/html/bp
el1-1.asp
```

Conventions

This section describes the conventions used in the text and code examples of this documentation set. It describes:

- Conventions in Text
- Conventions in Code Examples
- Conventions for Windows Operating Systems

Conventions in Text

We use various conventions in text to help you more quickly identify special terms. The following table describes those conventions and provides examples of their use.

Convention	Meaning	Example
Bold	Bold typeface indicates terms that are defined in the text or terms that appear in a glossary, or both.	When you specify this clause, you create an **index-organized table**.
Italics	Italic typeface indicates book titles or emphasis.	*Oracle Database Concepts* Ensure that the recovery catalog and target database do *not* reside on the same disk.
`UPPERCASE monospace (fixed-width) font`	Uppercase monospace typeface indicates elements supplied by the system. Such elements include parameters, privileges, datatypes, RMAN keywords, SQL keywords, SQL*Plus or utility commands, packages and methods, as well as system-supplied column names, database objects and structures, usernames, and roles.	You can specify this clause only for a `NUMBER` column. You can back up the database by using the `BACKUP` command. Query the `TABLE_NAME` column in the `USER_TABLES` data dictionary view. Use the `DBMS_STATS.GENERATE_STATS` procedure.
`lowercase monospace (fixed-width) font`	Lowercase monospace typeface indicates executables, filenames, directory names, and sample user-supplied elements. Such elements include computer and database names, net service names, and connect identifiers, as well as user-supplied database objects and structures, column names, packages and classes, usernames and roles, program units, and parameter values. **Note:** Some programmatic elements use a mixture of UPPERCASE and lowercase. Enter these elements as shown.	Enter `sqlplus` to start SQL*Plus. The password is specified in the `orapwd` file. Back up the datafiles and control files in the `/disk1/oracle/dbs` directory. The `department_id`, `department_name`, and `location_id` columns are in the `hr.departments` table. Set the `QUERY_REWRITE_ENABLED` initialization parameter to `true.` Connect as `oe` user. The `JRepUtil` class implements these methods.
`lowercase italic monospace (fixed-width) font`	Lowercase italic monospace font represents placeholders or variables.	You can specify the *`parallel_clause`*. Run *`old_release`*`.SQL` where *`old_release`* refers to the release you installed prior to upgrading.

Conventions in Code Examples

Code examples illustrate SQL, PL/SQL, SQL*Plus, or other command-line statements. They are displayed in a monospace (fixed-width) font and separated from normal text as shown in this example:

```
SELECT username FROM dba_users WHERE username = 'MIGRATE';
```

The following table describes typographic conventions used in code examples and provides examples of their use.

Convention	Meaning	Example
`[ ]`	Brackets enclose one or more optional items. Do not enter the brackets.	`DECIMAL (`*`digits`*` [ , `*`precision`*` ])`
`{ }`	Braces enclose two or more items, one of which is required. Do not enter the braces.	`{ENABLE | DISABLE}`
|	A vertical bar represents a choice of two or more options within brackets or braces. Enter one of the options. Do not enter the vertical bar.	`{ENABLE | DISABLE}` `[COMPRESS | NOCOMPRESS]`
`...`	Horizontal ellipsis points indicate either: ■ That we have omitted parts of the code that are not directly related to the example ■ That you can repeat a portion of the code	`CREATE TABLE ... AS `*`subquery`*`;` `SELECT `*`col1`*`, `*`col2`*`, ... , `*`coln`*` FROM employees;`
. . .	Vertical ellipsis points indicate that we have omitted several lines of code not directly related to the example.	`SQL> SELECT NAME FROM V$DATAFILE;` `NAME` `------------------------------------` `/fsl/dbs/tbs_01.dbf` `/fsl/dbs/tbs_02.dbf` `.` `.` `.` `/fsl/dbs/tbs_09.dbf` `9 rows selected.`
Other notation	You must enter symbols other than brackets, braces, vertical bars, and ellipsis points as shown.	`acctbal NUMBER(11,2);` `acct    CONSTANT NUMBER(4) := 3;`
`Italics`	Italicized text indicates placeholders or variables for which you must supply particular values.	`CONNECT SYSTEM/`*`system_password`* `DB_NAME = `*`database_name`*
`UPPERCASE`	Uppercase typeface indicates elements supplied by the system. We show these terms in uppercase in order to distinguish them from terms you define. Unless terms appear in brackets, enter them in the order and with the spelling shown. However, because these terms are not case sensitive, you can enter them in lowercase.	`SELECT last_name, employee_id FROM employees;` `SELECT * FROM USER_TABLES;` `DROP TABLE hr.employees;`

Convention	Meaning	Example

`lowercase`	Lowercase typeface indicates programmatic elements that you supply. For example, lowercase indicates names of tables, columns, or files.
	Note: Some programmatic elements use a mixture of UPPERCASE and lowercase. Enter these elements as shown.
		`SELECT last_name, employee_id FROM employees;`
		`sqlplus hr/hr`
		`CREATE USER mjones IDENTIFIED BY ty3MU9;`

Conventions for Windows Operating Systems

The following table describes conventions for Windows operating systems and provides examples of their use.

Convention	Meaning	Example
Choose Start >	How to start a program.	To start the Database Configuration Assistant, choose Start > Programs > Oracle - *HOME_NAME* > Configuration and Migration Tools > Database Configuration Assistant.
File and directory names	File and directory names are not case sensitive. The following special characters are not allowed: left angle bracket (<), right angle bracket (>), colon (:), double quotation marks ("), slash (/), pipe (|), and dash (-). The special character backslash (\) is treated as an element separator, even when it appears in quotes. If the file name begins with \\, then Windows assumes it uses the Universal Naming Convention.	`c:\winnt"\"system32 is the same as C:\WINNT\SYSTEM32`
`C:\>`	Represents the Windows command prompt of the current hard disk drive. The escape character in a command prompt is the caret (^). Your prompt reflects the subdirectory in which you are working. Referred to as the *command prompt* in this manual.	`C:\oracle\oradata>`

Special characters	The backslash (\) special character is sometimes required as an escape character for the double quotation mark (") special character at the Windows command prompt. Parentheses and the single quotation mark (') do not require an escape character. Refer to your Windows operating system documentation for more information on escape and special characters.	`C:\>exp scott/tiger TABLES=emp QUERY=\"WHERE job='SALESMAN' and sal<1600\"`
		`C:\>imp SYSTEM/`*`password`*` FROMUSER=scott TABLES=(emp, dept)`

`HOME_NAME`	Represents the Oracle home name. The home name can be up to 16 alphanumeric characters. The only special character allowed in the home name is the underscore.	`C:\> net start Oracle`*`HOME_NAME`*`TNSListener`

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"
- Chapter 3, "Building a Simple BPEL Process"

1

Introduction to Oracle BPEL Process Manager

This chapter provides a brief introduction to the Business Process Execution Language (BPEL), how Oracle BPEL Process Manager supports BPEL, and the types of BPEL designers available with Oracle BPEL Process Manager that enable you to design BPEL processes (JDeveloper BPEL Designer and Eclipse BPEL Designer). 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 the BPEL Designer?
- How to Use This Guide
- Tutorials and Demonstrations
- 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?

The Business Process Execution Language (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
- Design parallel flows of process execution
- Undo portions of processes when exceptions occur

See Also:

- `http://www.oracle.com/technology/bpel` for specific BPEL details, including links to BPEL specifications, white papers, product demonstrations, and discussion groups
- Chapter 4, "Manipulating XML Data in BPEL" through Chapter 13, "Interaction Patterns" for a review of key BPEL development concepts and code samples

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, WebSphere, 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 JDeveloper BPEL Designer:

- Transformations, workflows, worklists, notifications, and sensors
- Technology adapters (file, FTP, database, advanced queuing (AQ), and Java Messaging Service (JMS))

See Also:

- *Oracle Application Server Integration Installation Guide* for a list of supported operation systems
- *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 technology adapters

What Is the BPEL Designer?

Oracle BPEL Process Manager provides support for two types of BPEL designer environments for graphically designing BPEL processes:

- JDeveloper BPEL Designer
- Eclipse BPEL Designer

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 use wizards to integrate adapters and services such as workflows, transformations, notifications, sensors, and worklist task management with the process. Both BPEL designers can deploy the developed processes directly to Oracle BPEL Console. This facilitates the development and maintenance of BPEL processes.

JDeveloper BPEL Designer

JDeveloper BPEL Designer is integrated with Oracle JDeveloper 10*g*. Oracle JDeveloper 10*g* is an integrated development environment (IDE) for building applications and Web services using Java, XML, and SQL standards. Oracle JDeveloper 10*g* 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.

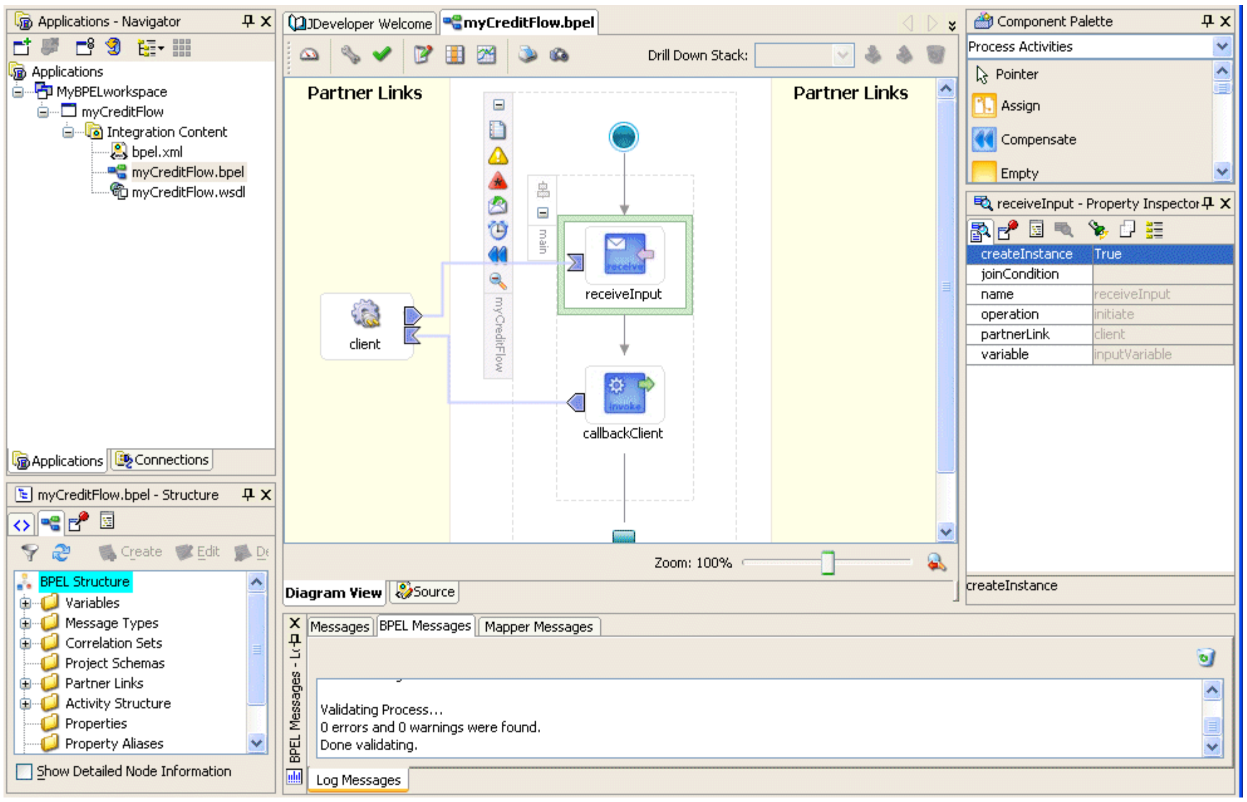
JDeveloper BPEL Designer uses BPEL as its native format. This means that processes built with JDeveloper BPEL Designer are 100% portable. JDeveloper BPEL Designer also enables you to view and modify the BPEL source without decreasing the usefulness of the tool.

Oracle BPEL Process Manager provides support for the following services and adapters in JDeveloper BPEL Designer:

- Transformations, workflows, worklists, notifications, and sensors
- Technology adapters (file, FTP, database, AQ, and JMS)

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

Figure 1–1 JDeveloper BPEL Designer



createInstance	True
joinCondition	
name	receiveInput
operation	initiate
partnerLink	client
variable	inputVariable

```
Validating Process...
0 errors and 0 warnings were found.
Done validating.
```

See Also:

- "Overview of JDeveloper BPEL Designer" on page 2-3 for a description of the sections of JDeveloper BPEL Designer shown in Figure 1–1
- *Oracle BPEL Process Manager Quick Start Guide* and *Oracle BPEL Process Manager Order Booking Tutorial* for tutorials that use JDeveloper BPEL Designer
- Online Help available from the **Help** main menu for additional details about Oracle JDeveloper 10*g*

Eclipse BPEL Designer

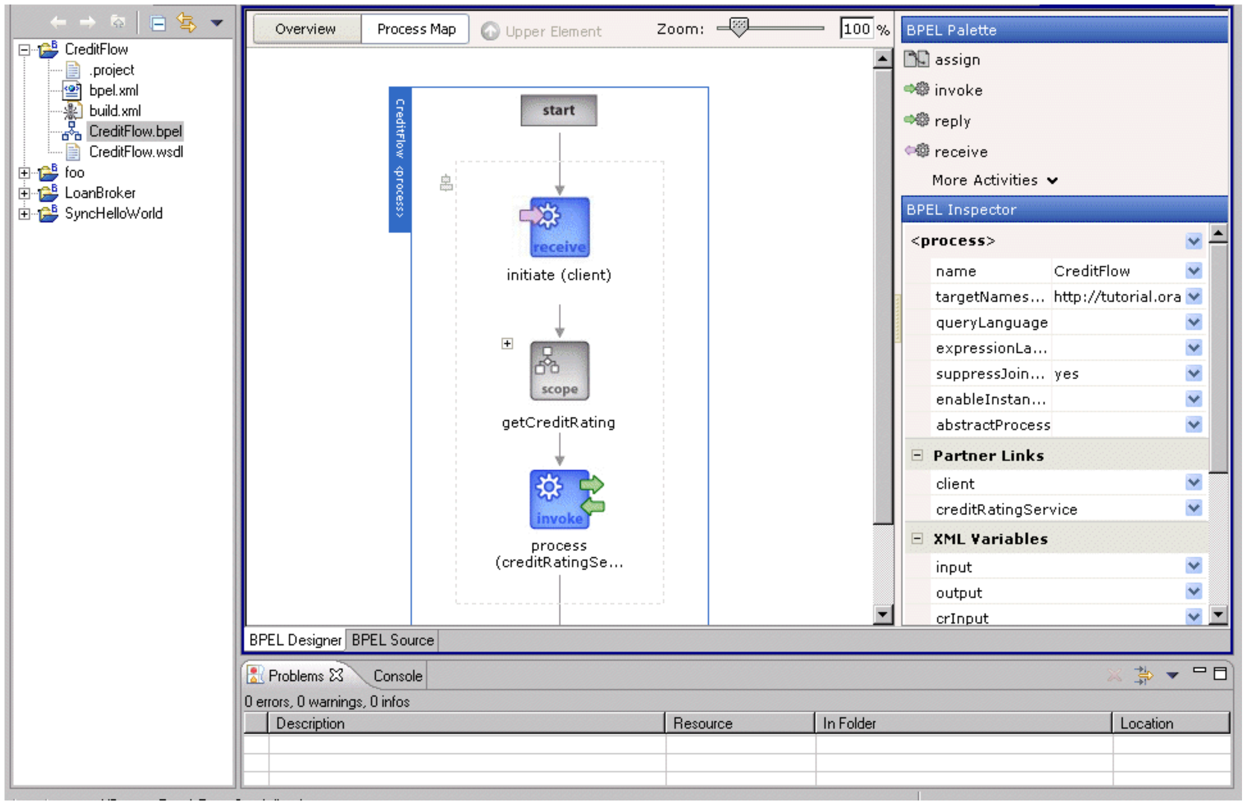
Eclipse BPEL Designer is integrated with the Eclipse platform. Eclipse is designed for building integrated development environments (IDEs) to create applications such as Web sites, embedded Java programs, C++ programs, and Enterprise JavaBeans. Eclipse provides you with mechanisms to use and rules to follow that lead to seamlessly integrated tools. These mechanisms are exposed through API interfaces, classes, and methods. Eclipse also provides useful building blocks and frameworks that facilitate developing new tools.

Eclipse BPEL Designer is a plug-in for the Eclipse 3.0 platform. Eclipse BPEL Designer uses standard BPEL, meaning that processes you design can be used with other BPEL servers (and vice-versa).

The JDeveloper BPEL Designer services and adapters described in "JDeveloper BPEL Designer" on page 1-3 are not currently available on Eclipse BPEL Designer.

Figure 1–2 shows Eclipse BPEL Designer with a BPEL process being designed.

Figure 1–2 Eclipse BPEL Designer



<process>

name	CreditFlow
targetNames...	http://tutorial.ora
queryLanguage	
expressionLa...	
suppressJoin...	yes
enableInstan...	
abstractProcess	

Partner Links: client, creditRatingService

XML Variables: input, output, crInput

0 errors, 0 warnings, 0 infos

Notes: Throughout this guide, the Eclipse BPEL Designer installation directory is referred to as `c:\orabpel`. If you have installed Eclipse BPEL Designer into a different directory, make the appropriate substitution throughout this guide.

See Also:

- "Overview of Eclipse BPEL Designer" on page 2-13 for a description of the sections of Eclipse BPEL Designer shown in Figure 1–2
- Chapter 3, "Building a Simple BPEL Process" for a tutorial that enables you to design and deploy a simple Hello World BPEL process
- `http://www.eclipse.org` for complete details about Eclipse
- `http://www.oracle.com/technology/bpel` for software download instructions and for details about running Oracle BPEL Process Manager and Eclipse BPEL Designer on Eclipse (including tutorials)

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

Part	Description
Part I, "Introduction and Concepts"	Chapters in this part provide an overview of the following topics: ■ BPEL specifications, Oracle BPEL Process Manager, and BPEL designers (JDeveloper BPEL Designer and Eclipse BPEL Designer) ■ Starting and stopping key Oracle BPEL Process Manager components ■ An introduction to JDeveloper BPEL Designer and Eclipse BPEL Designer, 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 Console for running a deployed BPEL processes ■ A Hello World BPEL process that you design and deploy in both JDeveloper BPEL Designer and Eclipse BPEL Designer
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 extensions to BPEL functionality to provide support for the following services: ■ Workflows ■ Transformations ■ Worklists ■ Notifications ■ Sensors
Part IV, "Development Life Cycle"	Chapters in this part describe how to run and manage deployed BPEL processes from Oracle BPEL Console.
Part V, "Reference Information"	Appendices in this part provide reference details about troubleshooting, supported activities, user accounts, XPath expression functions, and other issues.

See Also:

- *Oracle Adapters for Files, FTP, Databases, and Enterprise Messaging User's Guide* for specific details about configuring the file, FTP, database, AQ, and JMS adapters in a BPEL process
- *Oracle Application Server Adapter for Oracle Applications User's Guide* for information on using the Oracle Applications adapter

Tutorials and Demonstrations

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 tutorials, demonstrations, and references materials 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 `Oracle_Home\integration\orabpel\samples` directory for JDeveloper BPEL Designer and the `c:\orabpel\samples` directory for Eclipse BPEL Designer.

Table 1–2 describes the contents of the `samples` directory. If you are using JDeveloper BPEL Designer, you can also access details about this directory from the Start Menu by selecting **Start** > **All Programs** > **Oracle - *Oracle_Home*** > **Oracle BPEL Process Manager 10.1.2** > **Getting Started with Samples**.

Table 1–2 Tutorials, Demonstrations, and Reference Materials

Directory	Description
`demos`	Contains a set of common business scenarios and describes how they are implemented with BPEL. Table 1–3 on page 1-7 provides a specific description of the available demonstrations.
`interop`	Contains a set of BPEL projects showing the interoperability of Oracle BPEL Process Manager with Web services implemented with Microsoft .Net, Apache Axis, and BEA WebLogic
`references`	Contains a BPEL project for activities and concepts defined in the BPEL language. Table 1–4 on page 1-9 provides a specific description 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 a specific description of the available tutorials.
`utils`	Contains a set of building block services shared by the BPEL samples

Table 1–3 describes the BPEL process demonstrations available for use in the `demos` directory.

Table 1–3 demos Directory Contents

Directory	Description
`AmazonFlow`	Describes how to integrate an Amazon Web service with a BPEL process
`BankTransferDemo`	Describes how to perform a bank transfer
`CheckoutDemo`	Describes an interaction between a Java Server Page (JSP) user interface and a BPEL process

Table 1–3 (Cont.) demos Directory Contents

Directory	Description
`DocumentReview`	Describes how to create a business process for reviewing a document in parallel. A final reviewer reviews comments from each of the parallel reviewers. A worklist application views and acts on the tasks.
`GoogleFlow`	Describes how to invoke a Google Web service from a BPEL process
`HelpDeskServiceRequest`	Describes how to process a help desk service request. The demonstration uses a workflow for accepting or rejecting a service request.
`HotwireDemo`	Describes how to use correlation sets to create sophisticated and stateful interactions between a client and a BPEL process
`IBMSamples`	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
`LoanDemo`	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. Parts of this demonstration also appear in the *Oracle BPEL Process Manager Quick Start Guide*.
`LoanDemoPlus`	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
`LoanDemoPlusWithWorkflow`	Describes how to extend the LoanDemo sample to use a worklist application to approve or reject a loan application.
`OrderApproval`	Describes how to approve or reject an order.
`ParallelSearch`	Describes how to use Oracle BPEL Server to perform parallel synchronous invocations
`PaymentProcessor`	Describes how to use the following key concepts and features: ■ Use a pick activity to control a process flow based on the outcome of another process (including a timeout for the process). ■ Modify a process so that a parameter can be updated at run time without having to redeploy it.
`PriorityDemo`	Describes how to invoke a service *n* times using a `While` loop activity
`ResilientDemo`	Describes how to use a BPEL process to manage run time exceptions
`SalesforceFlow`	Describes how to integrate the Salesforce.com sForce Web services into a BPEL process (including authentication, session management, and dynamic load balancing)
`SleepBroker`	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
`TimeOffRequestDemo`	Describes how to design user interactions and workflow tasks within a BPEL process

Table 1–3 (Cont.) demos Directory Contents

Directory	Description
`VacationRequest`	Describes how to approve or reject a vacation request. The approval/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.
`XSLMapper`	Describes how to create a transformation that maps a purchase order schema to an invoice schema.

Table 1–4 describes the activity and concept code samples 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 1–4 references Directory Contents

Directory	Activity Description
`Assign`	Shows how to receive an input string, prefix `Hello` to it using an assign activity, and asynchronously return the result
`Catch`	Shows how an exception can be raised using the throw activity and managed using a catch activity
`CustomXPathFunction`	Shows how to use custom XPath functions within assign activities
`DynamicPartnerLink`	Shows how to update dynamic partner links
`Event`	Shows how to enable an asynchronous BPEL process and use event handlers to receive and process events while waiting for the asynchronous callback
`Flow`	Shows how to create parallel paths of execution within a BPEL process
`FlowN`	Shows how to receive an integer and create that number of branches
`Invoke`	Shows how to invoke a synchronous integer increment service
`JavaExec`	Shows how to use the BPEL exec extension to invoke a Java class from within a BPEL process
`Link`	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.
`Pick`	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.
`Receive`	Shows how to invoke an asynchronous loan service and wait for an asynchronous callback message using the BPEL receive activity
`Replay`	Shows how to replay an activity, such as a scope
`Reply`	Shows how to receive a string as input, perform an assign, and use the reply activity to synchronously return the modified string
`Switch`	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

Table 1–4 (Cont.) references Directory Contents

Directory	Activity Description
`Terminate`	Shows how to invoke a synchronous stock quoting service. The terminate activity then aborts, causing the final callback invoke activity to be skipped.
`Throw`	Shows how to throw a BPEL fault (without handling it) and cause the instance to fault.
`Wait`	Shows how to receive input, wait for 60 seconds, and asynchronously call back a client
`While`	Shows how to invoke an increment service *n* times with a while activity, where *n* is provided as an input value
`Xpath`	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.
`XPathFunction`	Shows how to define and use custom XPath functions within BPEL assign activities

See Also:

- Chapter 4, "Manipulating XML Data in BPEL" through Chapter 13, "Interaction Patterns" for activity development concepts and code samples
- Appendix C, "JDeveloper BPEL Designer Activities" for specific details about activities that you drag and drop in the BPEL designer

Table 1–5 describes the tutorials available for use in the `tutorials` directory.

Table 1–5 tutorials Directory Contents

Directory	Description
`101.HelloWorld`	This sample takes a string as input, appends `Hello` to it, and asynchronously generates a greeting as a response.
`102.InvokingProcesses`	This sample invokes a variety of processes, including JSPs and remote method invocations (RMIs).
`103.XMLDocuments`	This sample shows how to use XML variables and the assign activity to manipulate XML documents.
`104.SyncQuoteConsumer`	This sample shows how to use the invoke activity to invoke a synchronous stock quote service.
`105.AsyncCompositeLoanBroker`	This sample shows how to use the receive activity to receive a callback from an asynchronous loan processor Web service.
`106.ParallelFlows`	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.
`107.Exceptions`	This sample shows how to manage faults generated by invoke and throw activities.
`108.Timeouts`	This sample shows how to define and manage timeouts using the pick activity.

Table 1–5 (Cont.) tutorials Directory Contents

Directory	Description
`109.CorrelationSets`	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.
`110.UserTasks`	This sample shows how to integrate a user task within a process.
`111.CallingSessionBeans`	This sample shows how to use the BPEL exec extension to invoke a session bean from within a BPEL process.
`112.Arrays`	This sample shows how to design a BPEL process that uses arrays.
`113.ABCARouting`	This sample shows how to coordinate the flow of messages across three services: A, B, and C.
`114.XSLTTransformations`	This sample shows how to invoke XSLT transformations.
`116.SendEmails`	This sample shows how to send an e-mail in a BPEL process.
`117.ReceiveEmails`	This sample shows how to receive an e-mail in a BPEL process.
`118.JMSService`	This sample shows how to use JMS to select a JMS buyer process at Oracle BPEL Console and initiate a buyer flow. This sends a JMS message that is retrieved by a listener that outputs a message and sends a JMS response to the seller.
`119.JMSTopics`	This sample provides JMS samples for use with JBoss, OC4J, and WebLogic.
`120.XSQLExecution`	This sample shows how to execute XSQL.
`121.FileAdapter`	These samples show how to use the functionality of the file adapter.
`122.DBAdapter`	These samples show how to use the functionality of the database adapter.
`123.JMSAdapter`	These samples show how to use the functionality of the JMS adapter.
`124.AQAdapter`	These samples show how to use the functionality of the AQ adapter.
`125.ReportsSchema`	This sample shows how to build custom reports using the BPEL Process Manager reports schema. This sample highlights the use of sensors to track key milestones as part of process execution.
`126.DataAggregator`	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.
`127.OrderBookingTutorial`	This sample shows how to 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.
`128.GoogleFlow`	This sample shows a process that uses an external Web service to present information to the client. Processes designed with and without sensors are used.
`129.FTPAdapter`	These samples show how to use the functionality of the FTP adapter.
`130.SendEmailWithAttachments`	This sample shows how to send an e-mail with attachments.

Table 1–5 (Cont.) tutorials Directory Contents

Directory	Description
`701.LargeProcesses`	This sample shows how support is provided for processes with a large number of work items (10,000 or more).
`702.Bindings`	This sample shows how to: ■ Integrate Enterprise Java Beans (EJB) in a BPEL process ■ Call the HTTP `get` method from a BPEL process ■ Call a Java method from a BPEL process

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*

Summary

This chapter introduces BPEL, how Oracle BPEL Process Manager supports BPEL, and the types of BPEL designers available with Oracle BPEL Process Manager that enable you to design BPEL processes (JDeveloper BPEL Designer and Eclipse BPEL Designer). 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.

2 Getting Started with Oracle BPEL Process Manager

This chapter describes how to start key Oracle BPEL Process Manager components, including JDeveloper BPEL Designer, Eclipse BPEL Designer, Oracle BPEL Server, and Oracle BPEL Console. An overview of the main sections of JDeveloper BPEL Designer and Eclipse BPEL Designer 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.

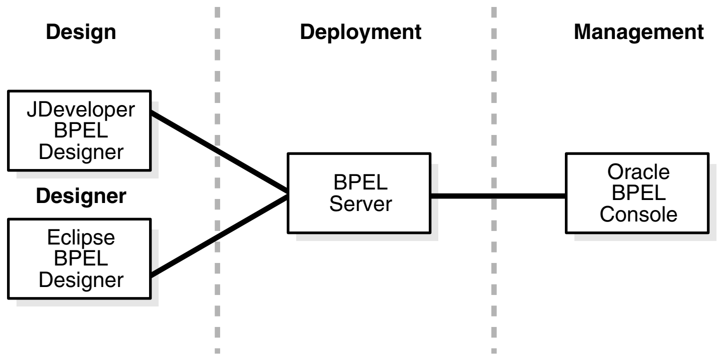
This chapter contains the following topics:

- Overview of Oracle BPEL Process Manager Components
- Starting Oracle BPEL Process Manager Components
- Overview of BPEL Designer Environments
- Overview of Activities
- Overview of Partner Links
- Overview of Oracle BPEL Server
- Overview of Oracle BPEL Console
- 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



Each component enables you to perform a specific set of tasks:

- The design environment (JDeveloper BPEL Designer or Eclipse BPEL Designer) 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 use wizards to integrate technology adapters and services such as workflows, worklists, transformations, notifications, and sensors 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 Console.

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

Starting Oracle BPEL Process Manager Components

As described in Chapter 1, "Introduction to Oracle BPEL Process Manager", Oracle BPEL Process Manager provides support for two types of BPEL designer environments for graphically designing and deploying BPEL processes:

- JDeveloper BPEL Designer
- Eclipse BPEL Designer

Follow these instructions to start Oracle BPEL Process Manager components on JDeveloper BPEL Designer and Eclipse BPEL Designer.

1. Select **Start** > **All Programs** > **Oracle -** ***Oracle_Home*** > **Oracle BPEL Process Manager** ***version_number*** > **Start BPEL PM Server** to start Oracle BPEL Server, which enables you to use the BPEL designer.
2. For JDeveloper BPEL Designer only, select **Start** > **All Programs** > **Oracle -** ***Oracle_Home*** > **Oracle BPEL Process Manager** ***version_number*** > **JDeveloper BPEL Designer**. The designer enables you to design and deploy BPEL processes graphically.

3. For Eclipse BPEL Designer only, select **Start** > **All Programs** > **Oracle - *Oracle_Home*** > **Oracle BPEL Process Manager *version_number*** > **BPEL Designer**. The designer enables you to design and deploy BPEL processes graphically.
4. Select **Start** > **All Programs** > **Oracle - *Oracle_Home*** > **Oracle BPEL Process Manager *version_number*** > **Developer Prompt** to open up a command prompt at the `Oracle_Home\integration\orabpel\samples` directory. This enables you to access tutorials, demonstrations, and reference materials and start any required tutorial Web services.
5. Select **Start** > **All Programs** > **Oracle - *Oracle_Home*** > **Oracle BPEL Process Manager *version_number*** > **BPEL Console** to start Oracle BPEL Console from which to run, monitor, and administer BPEL processes.
6. For JDeveloper BPEL Designer only, select **Start** > **All Programs** > **Oracle - *Oracle_Home*** > **Oracle BPEL Process Manager 10.1.2** > **Sample Worklist Application** to start the sample Oracle BPEL Worklist Application.
7. For JDeveloper BPEL Designer only, select **Start** > **All Programs** > **Oracle - *Oracle_Home*** > **Oracle BPEL Process Manager 10.1.2** > **Getting Started with Samples** for an overview of available tutorials, demonstrations, and reference materials in the `Oracle_Home\integration\orabpel\samples` directory.

Note: The version number that appears after Oracle BPEL Process Manager in the Start Menu sequence is determined by whether you are using JDeveloper BPEL Designer or Eclipse BPEL Designer.

See Also: `http://www.oracle.com/technology/bpel` for software download instructions and for details about running Oracle BPEL Process Manager and Eclipse BPEL Designer on Eclipse (including tutorials)

Overview of BPEL Designer Environments

This section provides an overview of the two types of BPEL designer environments for which Oracle BPEL Process Manager provides support:

- Overview of JDeveloper BPEL Designer
- Overview of Eclipse BPEL Designer

Overview of JDeveloper BPEL Designer

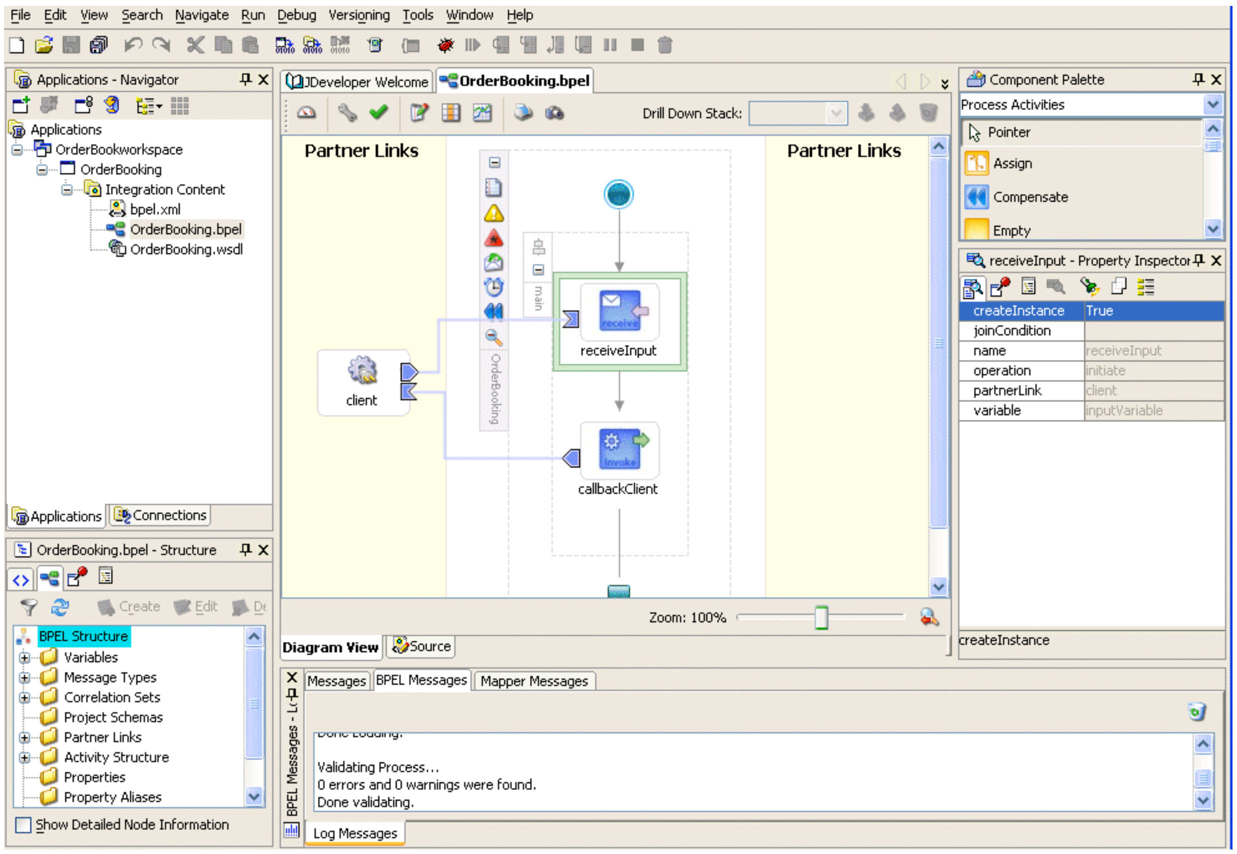
This section provides an overview of JDeveloper BPEL Designer. In this overview, you first create a workspace and a project. A workspace is a container in which to place projects. A project contains the BPEL process.

1. Create a workspace by selecting **File** > **New** > **Application Workspace** and providing the required details in the Create Application Workspace window (including selecting *not* to add a new empty project).
2. Right-click the newly created workspace and select **New Project**.
3. Double-click **BPEL Process Project** and provide the required details (including BPEL process name) in the BPEL Process Project window. A single project can contain only one BPEL process.

Note: Do not create a project name that begins with a number.

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

Figure 2–2 JDeveloper BPEL Designer Sections



createInstance	True
joinCondition	
name	receiveInput
operation	initiate
partnerLink	client
variable	inputVariable

```
Done Loading.
Validating Process...
0 errors and 0 warnings were found.
Done validating.
```

Each section of this view enables you to perform specific design and deployment tasks. Table 2–1 identifies the sections listed in Figure 2–2 and provides references to sections that describe their capabilities.

Table 2–1 JDeveloper BPEL Designer Sections

Section	Location in Figure 2–2	See Section
Applications Navigator	Upper left	"Applications Navigator" on page 2-5
Diagram View window and **Source** window	Middle	"Diagram View Window" on page 2-6 and "Source Window" on page 2-8
Process Activities selection of the **Component Palette**	Upper right	"Component Palette" on page 2-9
Property Inspector section	Lower right	"Property Inspector" on page 2-10

Table 2–1 (Cont.) JDeveloper BPEL Designer Sections

Section	Location in Figure 2–2	See Section
Structure Window	Lower left	"Structure Window" on page 2-11
Log Window	Bottom	"Log Window" on page 2-12

See Also: *Oracle BPEL Process Manager Quick Start Guide* and *Oracle BPEL Process Manager Order Booking Tutorial* for tutorials in which you create workspaces and projects

Applications 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 **FulfillOrder** 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 JDeveloper BPEL Designer (in this example, named **FulfillOrder** inside a workspace named **OrderBookworkspace**).

Figure 2–3 Applications Navigator

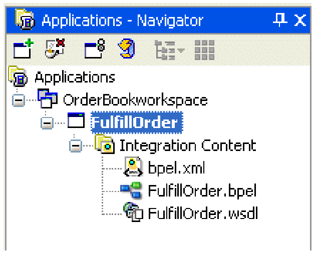


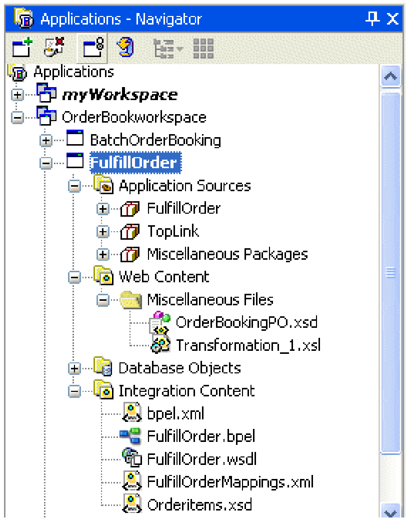
Table 2–2 describes these initial project files.

Table 2–2 Initial Project Files

File	Description
bpel.xml	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.
FulfillOrder.bpel	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.
FulfillOrder.wsdl	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.

As you design the project, additional files, folders, and elements can appear in the **Applications Navigator**. For example, Figure 2–4 shows the files that appear for a project in which you imported a schema (**OrderBookingPO.xsd**), configured the database adapter (under the **Application Sources** node), and created a **transform** activity (**Transformation_1.xsl** under the **Web Content** > **Miscellaneous Files** 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).

Figure 2–4 Applications Navigator (Expanded)



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

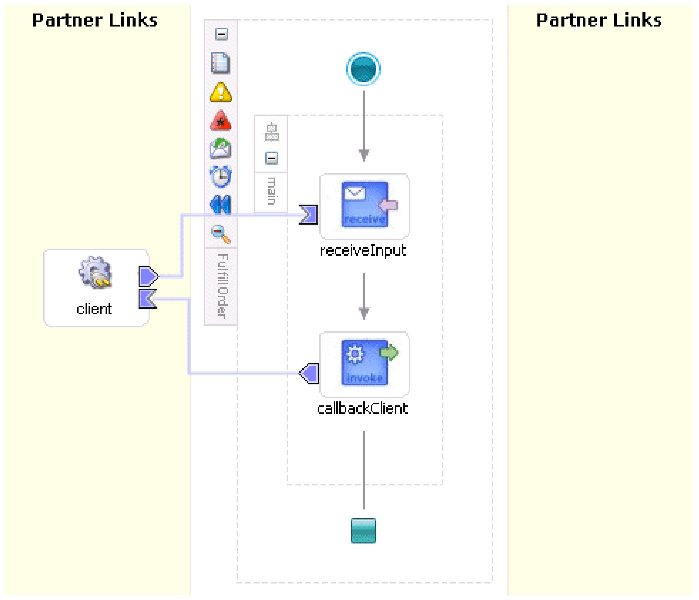
Diagram View Window

The **Diagram View** 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 **Applications Navigator**
- Click the **Diagram View** tab at the bottom of the window with the **.bpel** file selected

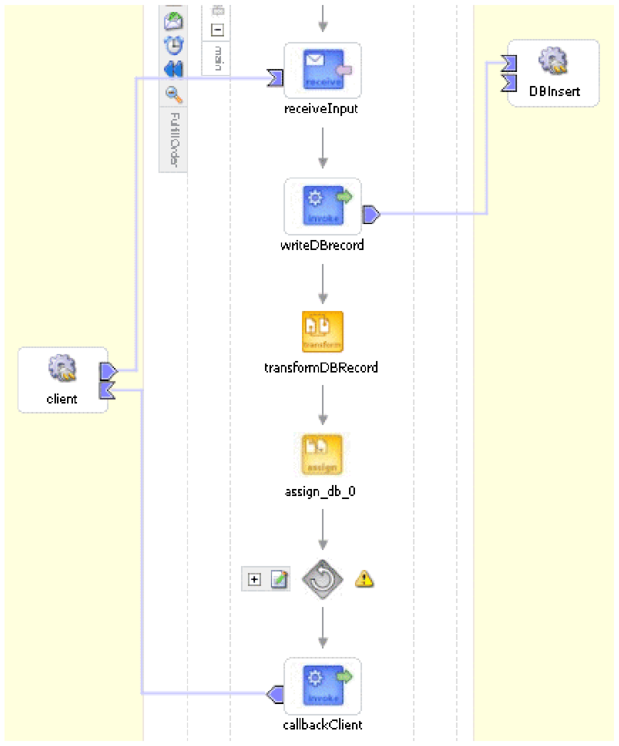
Figure 2–5 shows the activities automatically created with an asynchronous project. In the tutorials described in *Oracle BPEL Process Manager Quick Start Guide* and *Oracle BPEL Process Manager Order Booking Tutorial*, you add to the BPEL process by dragging and dropping activities, creating variables, creating partner links, and so on.

Figure 2–5 Diagram View (After Creation of an Asynchronous Project)



As you design the project by dragging and dropping activities, creating partner links, and so on, the **Diagram View** changes. Figure 2–6 shows the **Diagram View** later in the design phase after adding a partner link (in this example, named **DBInsert**) and the necessary activities (**invoke**, **receive**, **assign**, **transform**, and others).

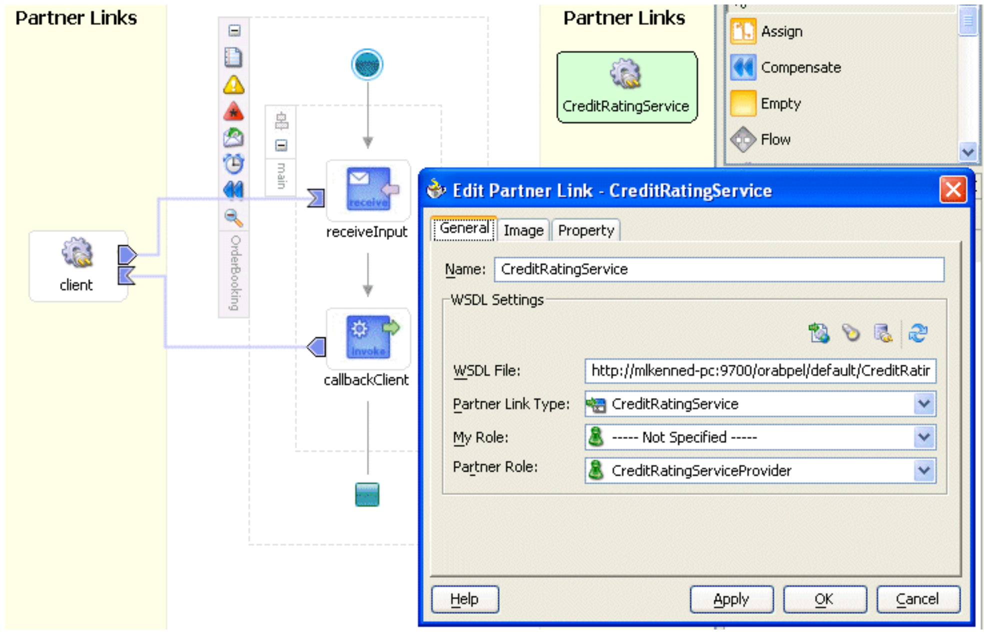
Figure 2–6 Diagram View (After Design Phase)



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 **CreditRatingService** partner link icon and property sheet that have just been created.

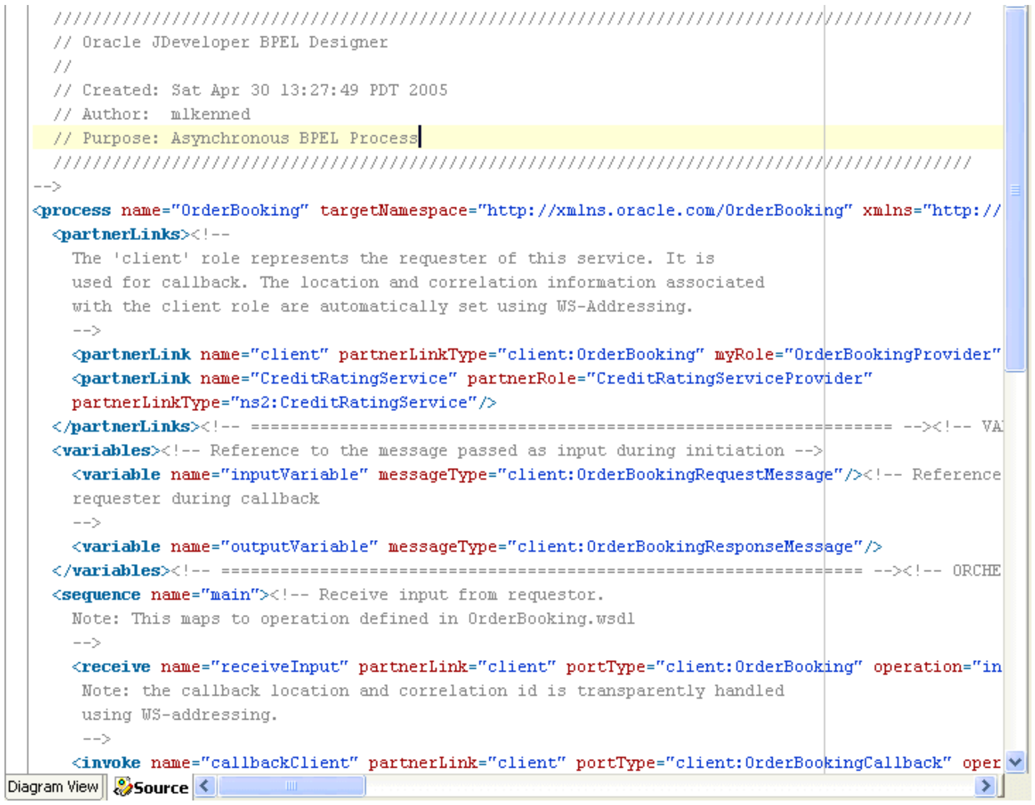
Figure 2–7



Name: CreditRatingService
WSDL File: http://mlkenned-pc:9700/orabpel/default/CreditRatir
Partner Link Type: CreditRatingService
My Role: ----- Not Specified -----
Partner Role: CreditRatingServiceProvider

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



```
  ////////////////////////////////////////////////////////////////////////////////////////////////////
  // Oracle JDeveloper BPEL Designer
  //
  // Created: Sat Apr 30 13:27:49 PDT 2005
  // Author:  mlkenned
  // Purpose: Asynchronous BPEL Process
  ////////////////////////////////////////////////////////////////////////////////////////////////////
-->
<process name="OrderBooking" targetNamespace="http://xmlns.oracle.com/OrderBooking" xmlns="http://
  <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="client:OrderBooking" myRole="OrderBookingProvider"
    <partnerLink name="CreditRatingService" partnerRole="CreditRatingServiceProvider"
    partnerLinkType="ns2:CreditRatingService"/>
  </partnerLinks><!-- ================================================================= --><!-- VA
  <variables><!-- Reference to the message passed as input during initiation -->
    <variable name="inputVariable" messageType="client:OrderBookingRequestMessage"/><!-- Reference
    requester during callback
    -->
    <variable name="outputVariable" messageType="client:OrderBookingResponseMessage"/>
  </variables><!-- ================================================================= --><!-- ORCHE
  <sequence name="main"><!-- Receive input from requestor.
    Note: This maps to operation defined in OrderBooking.wsdl
    -->
    <receive name="receiveInput" partnerLink="client" portType="client:OrderBooking" operation="in
     Note: the callback location and correlation id is transparently handled
     using WS-addressing.
     -->
    <invoke name="callbackClient" partnerLink="client" portType="client:OrderBookingCallback" oper
```

See Also: The following documentation for examples and descriptions of the types of syntax that appear in project files:

- Chapter 4, "Manipulating XML Data in BPEL" through Chapter 13, "Interaction Patterns"
- `Oracle_Home\integration\orabpel\samples` directory

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 View** of the BPEL process. The **Component Palette** displays only those pages relevant to the state of the **Diagram View**. **Process Activities** is 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 2–9 shows the **Process Activities** selection of the **Component Palette**.

Figure 2–9 Component Palette - Process Activities

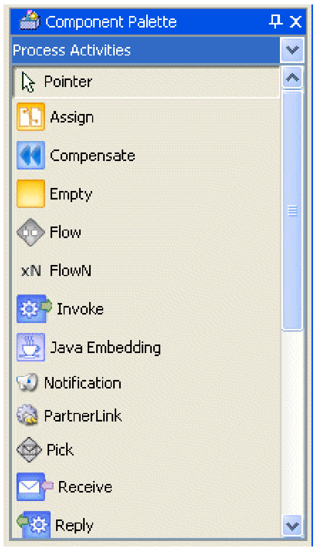
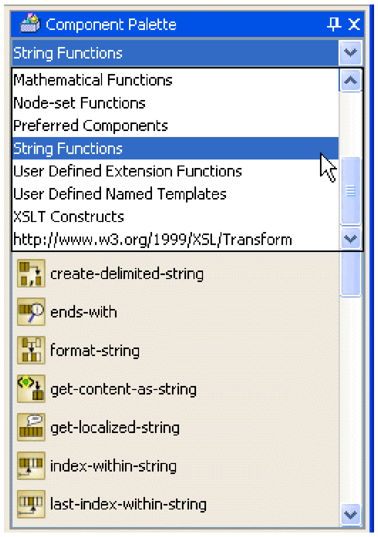


Figure 2–10 shows the **String Functions** category of the **Component Palette** that displays when you work in the transformation window of a **transform** activity.

Figure 2–10 Component Palette - Functions

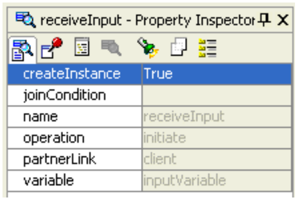


Note: If you want to learn more about the **Component Palette**, place the cursor in this section and press **F1** to display online Help.

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 View**. For example, single-clicking the **receiveInput receive** activity shown in Figure 2–2 on page 2-4 displays the information shown in Figure 2–11.

Figure 2–11 Property Inspector



createInstance	True
joinCondition	
name	receiveInput
operation	initiate
partnerLink	client
variable	inputVariable

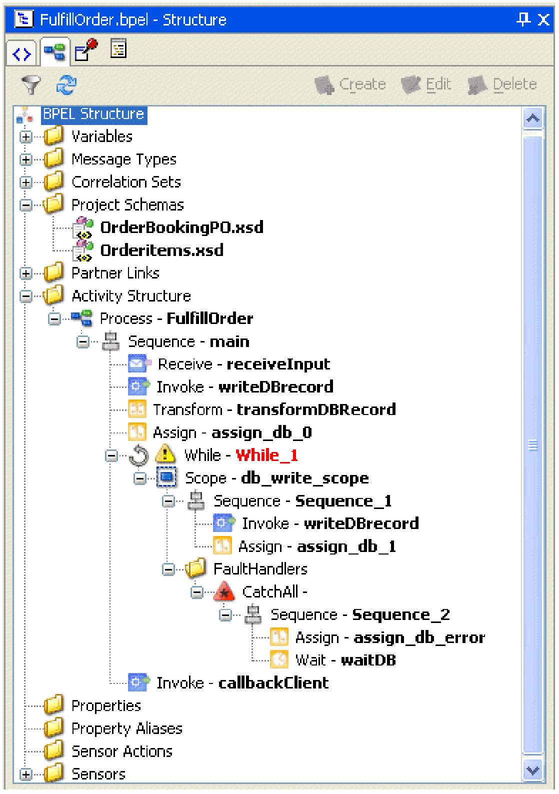
Structure Window

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 View**. 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 View**

Figure 2–12 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 View** for the **FulfillOrder** project shown in Figure 2–6 on page 2-7.

Figure 2–12 Structure Window (Expanded)



```
BPEL Structure
  Variables
  Message Types
  Correlation Sets
  Project Schemas
    OrderBookingPO.xsd
    Orderitems.xsd
  Partner Links
  Activity Structure
    Process - FulfillOrder
      Sequence - main
        Receive - receiveInput
        Invoke - writeDBrecord
        Transform - transformDBRecord
        Assign - assign_db_0
        While - While_1
          Scope - db_write_scope
            Sequence - Sequence_1
              Invoke - writeDBrecord
              Assign - assign_db_1
            FaultHandlers
              CatchAll -
                Sequence - Sequence_2
                  Assign - assign_db_error
                  Wait - waitDB
        Invoke - callbackClient
  Properties
  Property Aliases
  Sensor Actions
  Sensors
```

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

Log Window

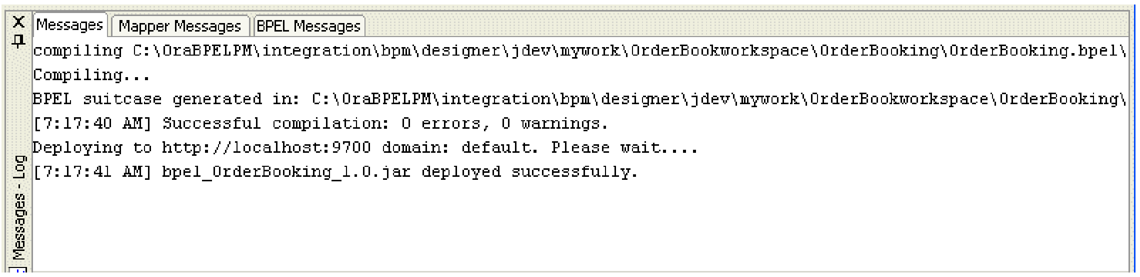
You validate, compile, and deploy a process by right-clicking the project name in the **Applications Navigator** and selecting **Deploy**. The **Log Window** shown at the bottom of Figure 2–2 then displays messages about the status of the deployment.

The following things must be true of a valid BPEL process:

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

Figure 2–13 shows a successful deployment message for a BPEL process. You can then run, monitor, and administer the process from Oracle BPEL Console.

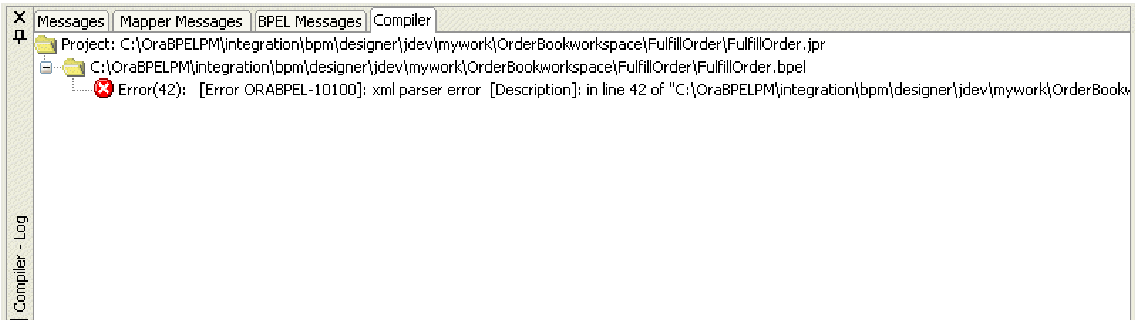
Figure 2–13 Successful Deployment Message



```
compiling C:\OraBPELPM\integration\bpm\designer\jdev\mywork\OrderBookworkspace\OrderBooking\OrderBooking.bpel\
Compiling...
BPEL suitcase generated in: C:\OraBPELPM\integration\bpm\designer\jdev\mywork\OrderBookworkspace\OrderBooking\
[7:17:40 AM] Successful compilation: 0 errors, 0 warnings.
Deploying to http://localhost:9700 domain: default. Please wait....
[7:17:41 AM] bpel_OrderBooking_1.0.jar deployed successfully.
```

If deployment is unsuccessful, messages appear that describe the type and location of the error, as shown in Figure 2–14. Double-click the error to navigate directly to the offending line in the source file referenced.

Figure 2–14 Unsuccessful Deployment Message



```
Project: C:\OraBPELPM\integration\bpm\designer\jdev\mywork\OrderBookworkspace\FulfillOrder\FulfillOrder.jpr
  C:\OraBPELPM\integration\bpm\designer\jdev\mywork\OrderBookworkspace\FulfillOrder\FulfillOrder.bpel
    Error(42):  [Error ORABPEL-10100]: xml parser error  [Description]: in line 42 of "C:\OraBPELPM\integration\bpm\designer\jdev\mywork\OrderBookw
```

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 Console" on page 2-21
- Chapter 19, "BPEL Process Deployment and Domain Management" for specific details about deploying and running BPEL processes

Overview of Eclipse BPEL Designer

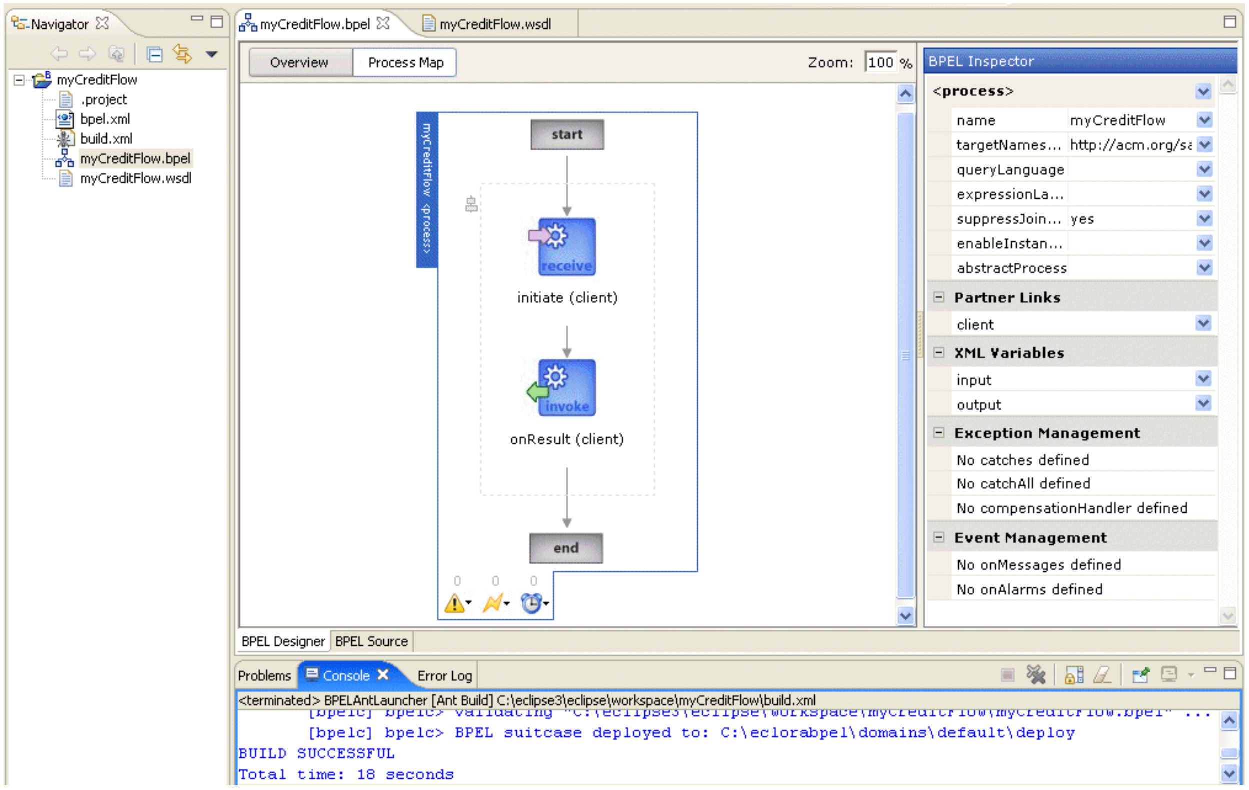
This section provides an overview of Eclipse BPEL Designer. Eclipse BPEL Designer is similar to JDeveloper BPEL Designer. Differences between the two BPEL designers are described. In this overview, you first create a project for the BPEL process.

1. Select **File** > **New** > **Project** to display the New Project window.
2. Select **Oracle BPEL Project** (the default selection) and click **Next** to display the Create a BPEL Project window.
3. Provide the required details (including BPEL process name) and click **Finish**. A single project can contain only one BPEL process.

Note: Do not create a project name that begins with a number.

After you create the project, Eclipse BPEL Designer displays the sections shown in Figure 2–15. You can also access this view by double-clicking the **.bpel** file of the project and selecting **Process Map**. In this example, the project is named **myCreditFlow**.

Figure 2–15 Eclipse BPEL Designer Sections



BPEL Inspector

<process>

name	myCreditFlow
targetNames…	http://acm.org/sa
queryLanguage	
expressionLa…	
suppressJoin…	yes
enableInstan…	
abstractProcess	

Partner Links
- client

XML Variables
- input
- output

Exception Management
- No catches defined
- No catchAll defined
- No compensationHandler defined

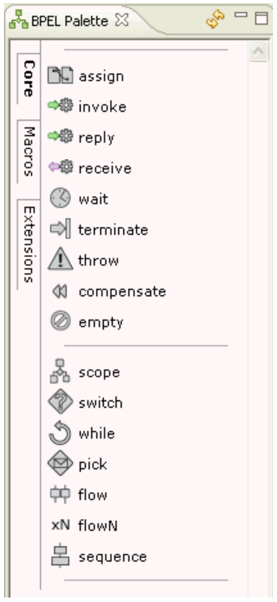
Event Management
- No onMessages defined
- No onAlarms defined

```
<terminated> BPELAntLauncher [Ant Build] C:\eclipse3\eclipse\workspace\myCreditFlow\build.xml
       [bpelc] bpelc> validating "C:\eclipse3\eclipse\workspace\myCreditFlow\myCreditFlow.bpel" ...
       [bpelc] bpelc> BPEL suitcase deployed to: C:\eclorabpel\domains\default\deploy
BUILD SUCCESSFUL
Total time: 18 seconds
```

In addition, the **BPEL Palette** of activities displays on this page. If the **BPEL Palette** does not currently display, perform the following steps:

1. Select **Show View** > **Other** from the **Windows** main menu
2. Select **BPEL** > **BPEL Palette** on the Show View window, and click **OK** to display the **BPEL Palette** in Eclipse BPEL Designer.

Figure 2–16 BPEL Palette



Each section of this view enables you to perform specific design and deployment tasks. Table 2–3 identifies the sections listed in Figure 2–15 and Figure 2–16 and provides references to sections that describe their capabilities.

Table 2–3 Eclipse BPEL Designer Sections

Section	Location	See Section
Navigator	Upper left of Figure 2–15	"Navigator" on page 2-15
Process Map window, **Overview** Window, and **BPEL Source** window	Middle of Figure 2–15	"Process Map and Overview Windows" on page 2-15 and "BPEL Source Window" on page 2-17
BPEL Inspector	Left side of Figure 2–15	"BPEL Inspector" on page 2-17
Log Window	Bottom of Figure 2–15	"Log Window" on page 2-17
BPEL Palette	Figure 2–16	"BPEL Palette" on page 2-18

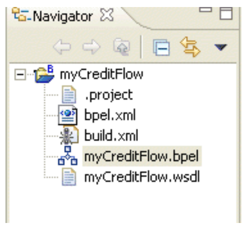
See Also:

- Chapter 3, "Building a Simple BPEL Process" for a tutorial that enables you to design and deploy a simple Hello World BPEL process
- `http://www.eclipse.org` for complete details about graphically designing processes in the Eclipse environment
- `http://www.oracle.com/technology/bpel` for details about software downloads and running Oracle BPEL Process Manager and Eclipse BPEL Designer on Eclipse (including tutorials)

Navigator

The **Navigator** shown in the upper left part of Figure 2–15 displays the project files in Eclipse BPEL Designer. See Table 2–2 on page 2-5 for descriptions of the files that initially appear when you first create a project. Eclipse BPEL Designer also includes an additional project file: **build.xml**. This is an Apache Ant file used for compiling and deploying processes in Eclipse BPEL Designer.

Figure 2–17 Navigator



Process Map and Overview Windows

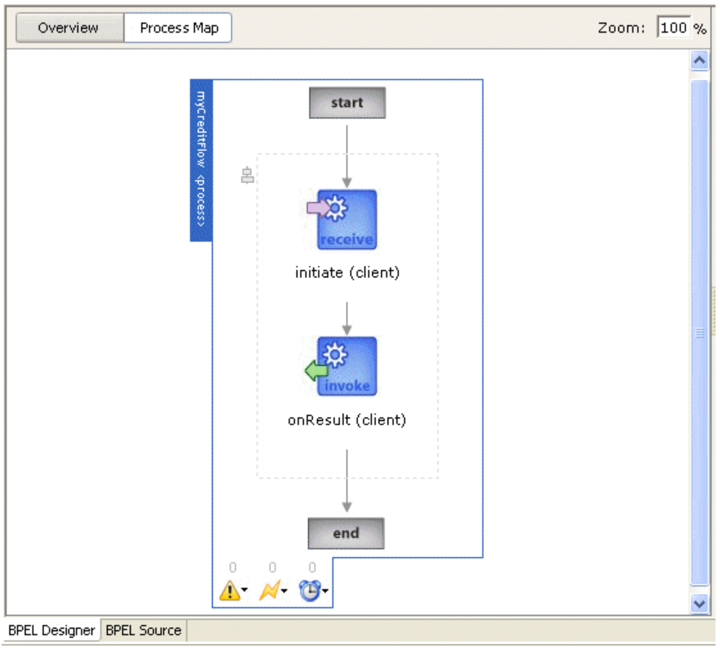
The **Process Map** shown in the middle of Figure 2–15 provides a visual view of the BPEL process that you design. The **Process Map** is similar to the **Diagram View** window in JDeveloper BPEL Designer. This view displays when you perform one of the following actions:

- Double-click the **.bpel** file name in the **Applications Navigator**

- Click the **BPEL Designer** tab; then click the **Process Map** tab

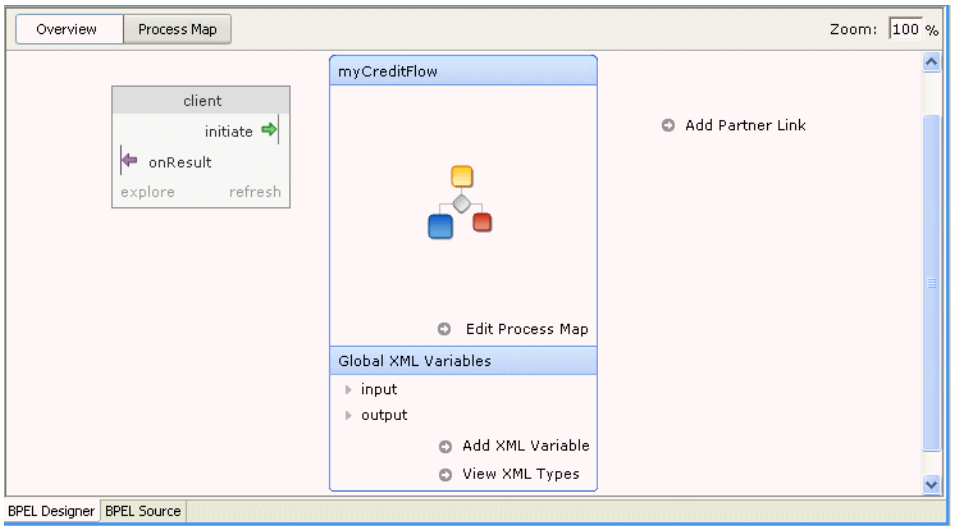
Figure 2–18 shows the activities automatically created with an asynchronous project. You add to the process by dragging and dropping activities, creating variables, creating partner links, and so on.

Figure 2–18 Process Map Window



Click **Overview** to display an overview on the BPEL process, as shown in Figure 2–19. You can perform tasks such as adding partner links and creating variables in the **Overview** window.

Figure 2–19 Overview Window



See Also: The following tutorials that use Eclipse BPEL Designer to design BPEL processes:

- Chapter 3, "Building a Simple BPEL Process" for instructions on building a Hello World tutorial
- `http://www.oracle.com/technology/bpel` for links to tutorials that introduce key BPEL concepts

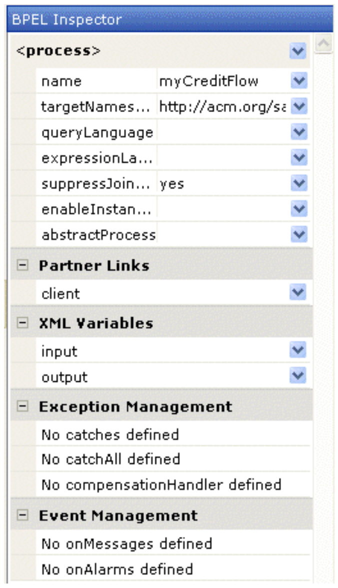
BPEL Source Window

The **BPEL Source** window is similar to that in JDeveloper BPEL Designer. Click **Source Window** to view the syntax inside the project files, which is similar to that shown for JDeveloper BPEL Designer in Figure 2–8 on page 2-9.

BPEL Inspector

The **BPEL Inspector** enables you to perform tasks such as editing existing activities, adding copy rules to an **assign** activity, creating global variables, creating partner links, and so on. Figure 2–20 shows the **BPEL Inspector**.

Figure 2–20 BPEL Inspector



BPEL Inspector

<process>

name	myCreditFlow
targetNames...	http://acm.org/sa
queryLanguage	
expressionLa...	
suppressJoin...	yes
enableInstan...	
abstractProcess	

Partner Links

client

XML Variables

input

output

Exception Management

No catches defined

No catchAll defined

No compensationHandler defined

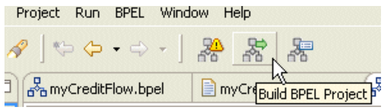
Event Management

No onMessages defined

No onAlarms defined

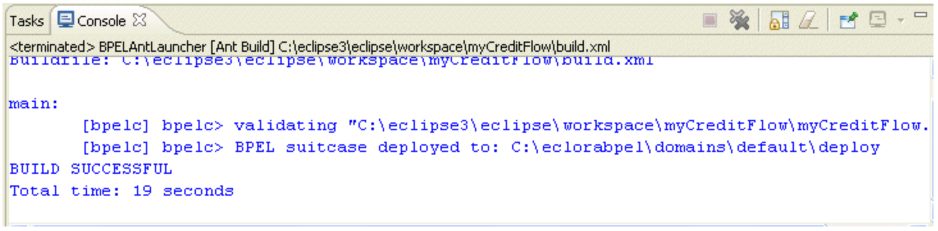
Log Window

You validate, compile, and deploy a process by clicking the **Build BPEL Project** icon in the tool bar.



The **Log Window** at the bottom of Eclipse BPEL Designer then displays messages about the status of the deployment. Figure 2–21 shows a successful deployment message for a BPEL process. You can then run, monitor, and administer the process from Oracle BPEL Console.

Figure 2–21 Log Window



```
<terminated> BPELAntLauncher [Ant Build] C:\eclipse3\eclipse\workspace\myCreditFlow\build.xml
Buildfile: C:\eclipse3\eclipse\workspace\myCreditFlow\build.xml

main:
       [bpelc] bpelc> validating "C:\eclipse3\eclipse\workspace\myCreditFlow\myCreditFlow.
       [bpelc] bpelc> BPEL suitcase deployed to: C:\eclorabpel\domains\default\deploy
BUILD SUCCESSFUL
Total time: 19 seconds
```

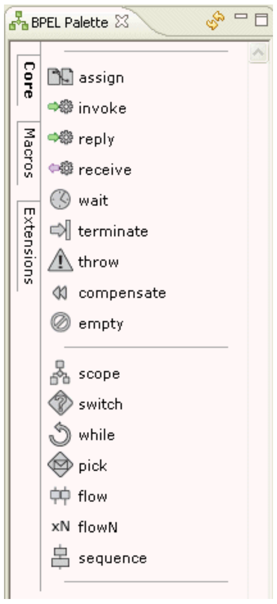
See Also:

- "Overview of Oracle BPEL Console" on page 2-21
- Chapter 19, "BPEL Process Deployment and Domain Management" for specific details about deploying and running BPEL processes

BPEL Palette

The **BPEL Palette** is similar to the **Process Activities** selection of the **Component Palette** in JDeveloper BPEL Designer. The **BPEL Palette** displays a set of activities that you drag and drop into the **Process Map** of the BPEL process. The set of activities available with Eclipse BPEL Designer differs slightly from those available with JDeveloper BPEL Designer. For example, unlike JDeveloper BPEL Designer, a partner link is not listed as a BPEL Palette activity in Eclipse BPEL Designer. JDeveloper BPEL Designer also includes additional activities for transformations, user tasks (workflows), and notifications.

Figure 2–22 BPEL Palette



See Also: "Overview of Oracle BPEL Process Manager Services" on page 2-22

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. JDeveloper BPEL Designer and Eclipse BPEL Designer both include 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 2-7 provides an example of this design process. Activities enable you to perform specific tasks within a process. For example:

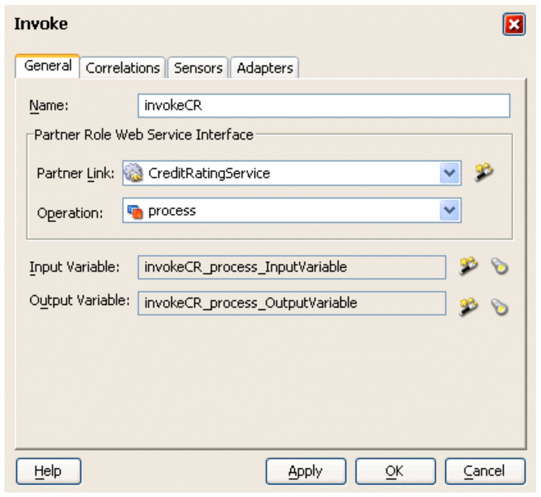
- An assign activity enables you to manipulate data, such as copying the contents of one variable to another.

- 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 2–23 shows an example of a property window (for this example, an invoke activity). In this example, you invoke a partner link named **CreditRatingService** and define its attributes.

Figure 2–23 Invoke Activity Example



Name: invokeCR
Partner Role Web Service Interface
Partner Link: CreditRatingService
Operation: process
Input Variable: invokeCR_process_InputVariable
Output Variable: invokeCR_process_OutputVariable

See Also:

- Appendix C, "JDeveloper BPEL Designer Activities" for descriptions of available activities
- Part II, "Reviewing Key BPEL Development Concepts and Code Samples" for activity concepts and code examples
- `*Oracle_Home*\integration\orabpel\samples directory\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

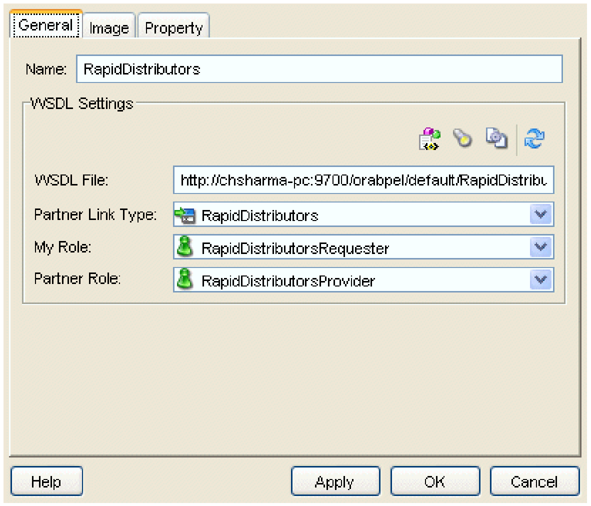
The term *partner link* has also been mentioned frequently in both Chapter 1, "Introduction to Oracle BPEL Process Manager" and in this chapter. A partner link enables you to define the external services with which the BPEL process is to interact. Figure 2–24 shows the partner link icon (in this example, named **CreditRatingService**).

Figure 2–24 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 2–6 on page 2-7 shows an example of a partner link named **DBInsert** being invoked by a BPEL process.

Figure 2–25 shows an example of the attributes of a partner link for a service named **RapidDistributors**.

Figure 2–25 PartnerLink Window



Name: RapidDistributors

WSDL Settings

WSDL File: http://chsharma-pc:9700/orabpel/default/RapidDistribu

Partner Link Type: RapidDistributors

My Role: RapidDistributorsRequester

Partner Role: RapidDistributorsProvider

Table 2–4 describes the fields of the PartnerLink window.

Table 2–4 PartnerLink Window Fields

Field	Description
Name	A unique and recognizable name you provide for the partner link.
WSDL File	The name and location of the Web Services Description Language (WSDL) file that you select for the partner link. Click the **flashlight** icon (second icon from the left above the **WSDL File** field) to access a window for selecting the WSDL file to use.
Partner Link Type	The partner link defined in the WSDL file.
My Role	The role performed by the BPEL process. In this case, the BPEL process is the requester.
Partner Type	The role performed by the partner link (in this example, the **RapidDistributors** service). In this case, **RapidDistributors** is the provider.

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 Console.

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.

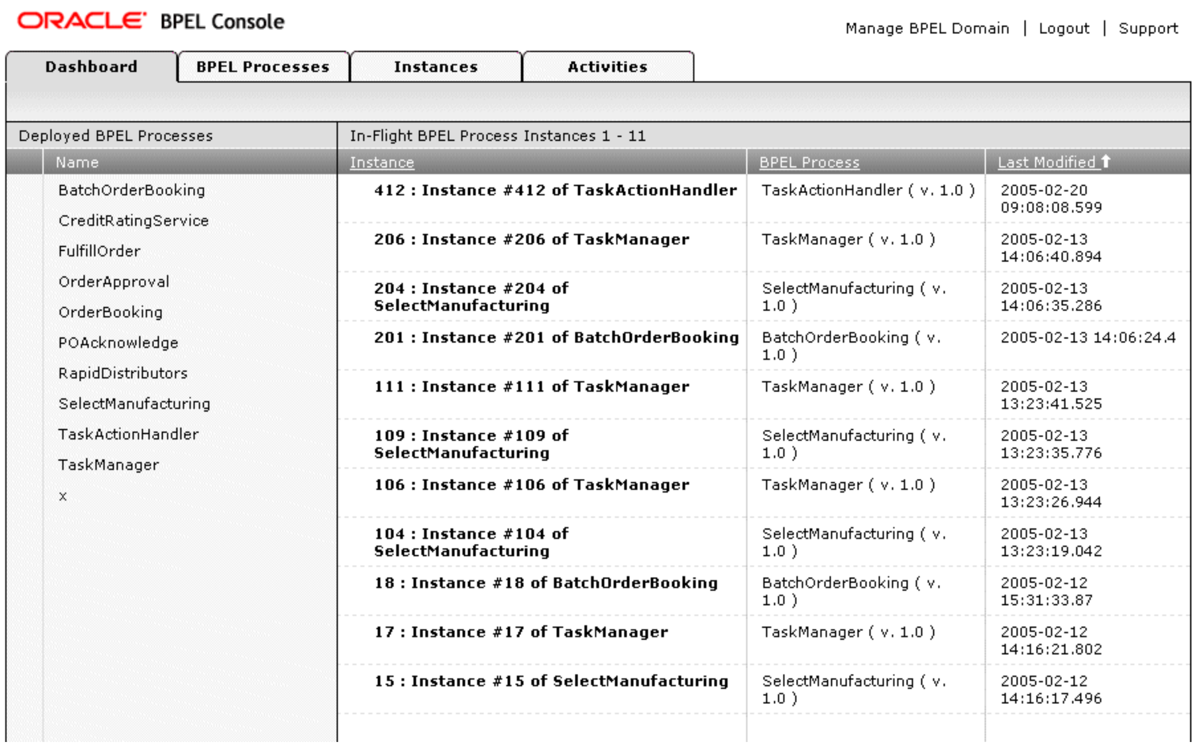
See Also: Chapter 19, "BPEL Process Deployment and Domain Management"

Overview of Oracle BPEL Console

Oracle BPEL Console enables you to run, monitor, and administer BPEL processes designed and deployed with either JDeveloper BPEL Designer or Eclipse BPEL Designer. You can also manage BPEL domains from Oracle BPEL Console. Access Oracle BPEL Console by selecting **Start** > **All Programs** > **Oracle -** ***Oracle_Home*** > **Oracle BPEL Process Manager** ***version_number*** > **BPEL Console**.

Figure 2–26 shows the main page of Oracle BPEL Console. In this example, a number of deployed BPEL processes and external services appear in the **Dashboard** tab.

Figure 2–26 Oracle BPEL Console



Deployed BPEL Processes

Name
BatchOrderBooking
CreditRatingService
FulfillOrder
OrderApproval
OrderBooking
POAcknowledge
RapidDistributors
SelectManufacturing
TaskActionHandler
TaskManager
x

In-Flight BPEL Process Instances 1 - 11

Instance	BPEL Process	Last Modified
412 : Instance #412 of TaskActionHandler	TaskActionHandler (v. 1.0)	2005-02-20 09:08:08.599
206 : Instance #206 of TaskManager	TaskManager (v. 1.0)	2005-02-13 14:06:40.894
204 : Instance #204 of SelectManufacturing	SelectManufacturing (v. 1.0)	2005-02-13 14:06:35.286
201 : Instance #201 of BatchOrderBooking	BatchOrderBooking (v. 1.0)	2005-02-13 14:06:24.4
111 : Instance #111 of TaskManager	TaskManager (v. 1.0)	2005-02-13 13:23:41.525
109 : Instance #109 of SelectManufacturing	SelectManufacturing (v. 1.0)	2005-02-13 13:23:35.776
106 : Instance #106 of TaskManager	TaskManager (v. 1.0)	2005-02-13 13:23:26.944
104 : Instance #104 of SelectManufacturing	SelectManufacturing (v. 1.0)	2005-02-13 13:23:19.042
18 : Instance #18 of BatchOrderBooking	BatchOrderBooking (v. 1.0)	2005-02-12 15:31:33.87
17 : Instance #17 of TaskManager	TaskManager (v. 1.0)	2005-02-12 14:16:21.802
15 : Instance #15 of SelectManufacturing	SelectManufacturing (v. 1.0)	2005-02-12 14:16:17.496

See Also:

- "Testing the BPEL Process" on page 3-17 for an Eclipse BPEL Designer tutorial in which you run a deployed process from Oracle BPEL Console
- Chapter 19, "BPEL Process Deployment and Domain Management" for specific details about running a deployed process from Oracle BPEL Console
- *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 JDeveloper BPEL Designer 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

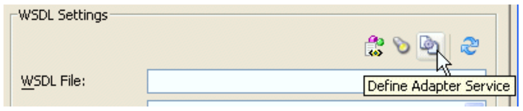
Types	Description	See Section
Transformations	A transform activity is provided that enables you to create transformations that map source data to target data. For example, you can map incoming purchase order source data into outgoing purchase order acknowledgment target data.	Chapter 14, "XSLT Mapper and Transformations" "Transform Activity" on page C-17
Notifications	A notification activity enables you to send notification about an event to a user, group, or destination address. You can send a notification by e-mail, voice mail, or short message service (SMS).	Chapter 15, "Oracle BPEL Process Manager Notification Service" "Notification Activity" on page C-10
Workflows	Workflow enables you to integrate systems and services with human workflow into a single process flow. A user task activity is provided that invokes the Workflow wizard. This enables you to: ■ Select a workflow pattern to use ■ Specify workflow details ■ Specify a task outcome (such as accept or reject) ■ Specify the notification types (for example, if a task completes, errors out, or expires, and which recipients are notified of this action) ■ Specify the initial assignee for a task The criteria that you define with the Workflow wizard enables you to use the Oracle BPEL Worklist Application when you run the BPEL process.	Chapter 16, "Oracle BPEL Process Manager Workflow Services" "User Task" on page C-18
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 17, "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 18, "Sensors"

See Also: *Oracle BPEL Process Manager Order Booking Tutorial* for tutorials that describe how to design BPEL processes that use the services described in Table 2–5

Overview of Oracle BPEL Process Manager Technology Adapters

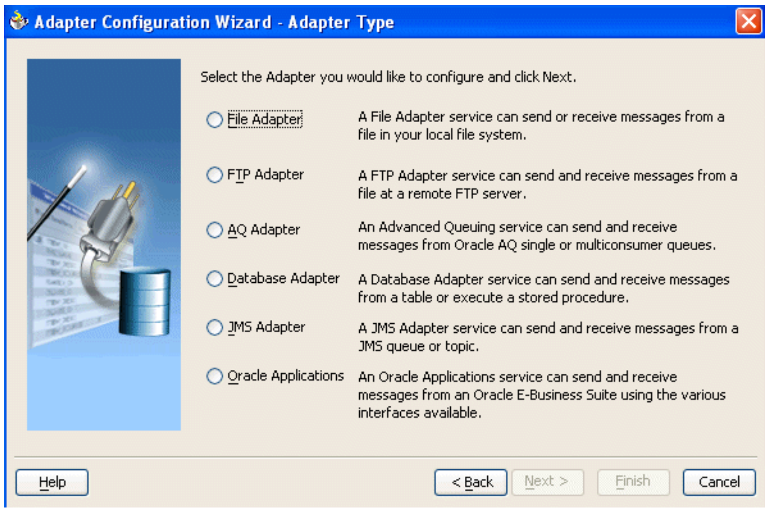
The Partner Link Window shown in Figure 2–25 on page 2-20 also enables you to take advantage of another key feature that Oracle BPEL Process Manager and JDeveloper BPEL Designer provide. Click the **Define Adapter Service** icon shown in Figure 2–27 to access the Adapter Configuration wizard.

Figure 2–27 Defining an Adapter



Adapters enable you to integrate the BPEL processes with access to file systems, database tables, database queues, Java Message Services (JMS), and Oracle E-Business Suite. This wizard enables you to configure the types of adapters shown in Figure 2–28 for use with the BPEL process:

Figure 2–28 Adapter Types

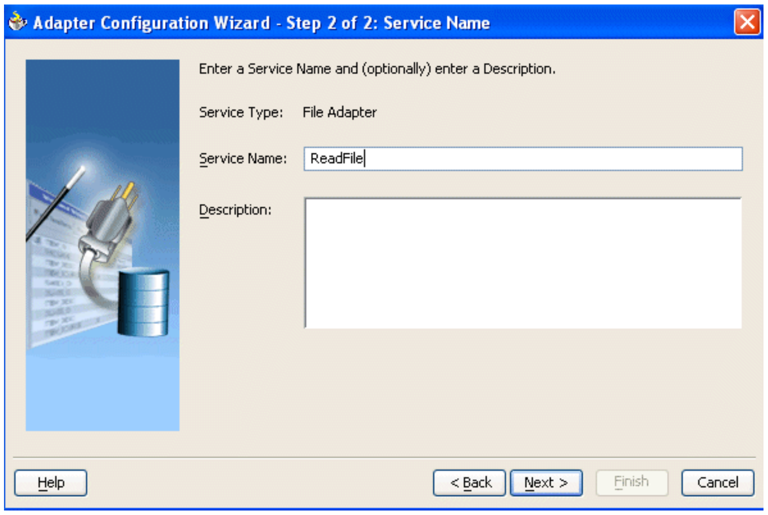


Select the Adapter you would like to configure and click Next.

File Adapter	A File Adapter service can send or receive messages from a file in your local file system.
FTP Adapter	A FTP Adapter service can send and receive messages from a file at a remote FTP server.
AQ Adapter	An Advanced Queuing service can send and receive messages from Oracle AQ single or multiconsumer queues.
Database Adapter	A Database Adapter service can send and receive messages from a table or execute a stored procedure.
JMS Adapter	A JMS Adapter service can send and receive messages from a JMS queue or topic.
Oracle Applications	An Oracle Applications service can send and receive messages from an Oracle E-Business Suite using the various interfaces available.

When you select an adapter type, the Service Name window shown in Figure 2–29 prompts you to enter a name. For this example, **File Adapter** was selected in Figure 2–28. When the wizard completes, a WSDL file by this service name appears in the **Applications 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 **Applications Navigator**.

Figure 2–29 Adapter Service Name



Enter a Service Name and (optionally) enter a Description.

Service Type: File Adapter

Service Name: ReadFile

Description:

The Adapter Configuration wizard windows that appear after the Service Name window are based on the adapter type you selected. These configuration windows and the information you must provide are described in later chapters of this guide.

See Also:

- *Oracle Adapters for Files, FTP, Databases, and Enterprise Messaging User's Guide* for specific details about configuring the file, FTP, database, AQ, and JMS adapters in a BPEL process
- *Oracle Application Server Adapter for Oracle Applications User's Guide* for information on using the Oracle Applications adapter
- "PartnerLink Activity" on page C-10
- *Oracle BPEL Process Manager Order Booking Tutorial* for tutorials that describe how to design BPEL processes that use the database adapter and file functionality of the file adapter

Summary

This chapter describes how to start key Oracle BPEL Process Manager components, including JDeveloper BPEL Designer, Eclipse BPEL Designer, Oracle BPEL Server, and Oracle BPEL Console. An overview of the main sections of JDeveloper BPEL Designer and Eclipse BPEL Designer 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.

3 Building a Simple BPEL Process

In this chapter, you create, build, deploy, and test a a simple BPEL process: a synchronous Hello World application. It accepts a name as an input message and returns the message *Hello* followed by the name through a synchronous reply. You will design the process in Eclipse BPEL Designer, but alternatively, you can use JDeveloper BPEL Designer, which provides similar functionality.

This chapter contains the following topics:

- Overview of Building a Simple BPEL Process
- Creating a New BPEL Project Using Eclipse BPEL Designer
- Browsing a New Project
- Viewing the WSDL Interface of a BPEL Process
- Switching Between the Overview, Process Map, and Source Code
- Reviewing the BPEL Source Code
- Adding an Assign Activity to the Process Map
- Compiling and Deploying the BPEL Process
- Testing the BPEL Process
- Summary

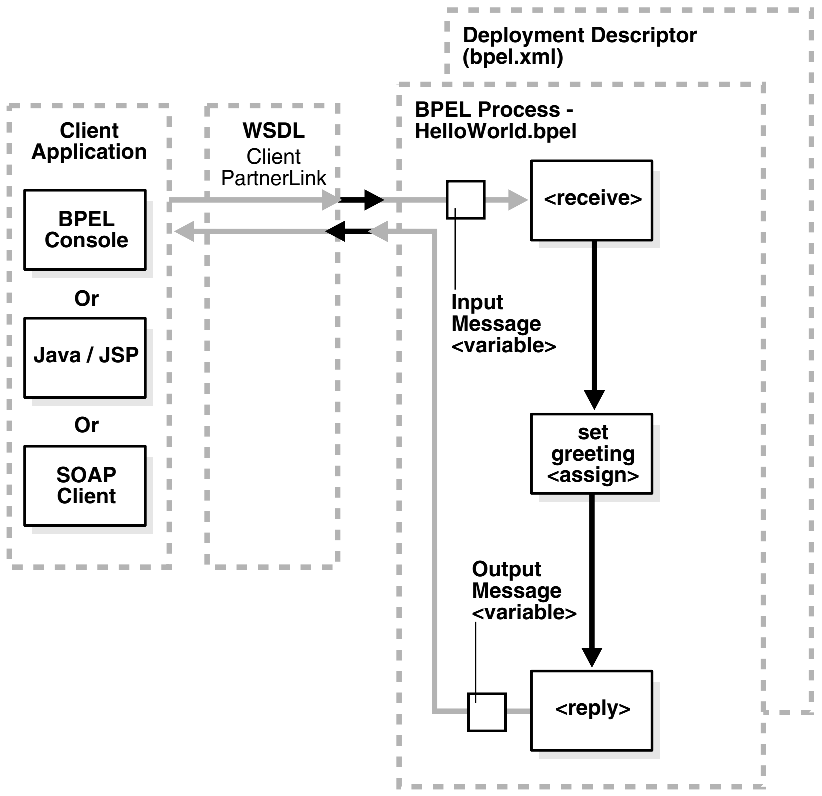
Overview of Building a Simple BPEL Process

The Hello World application is familiar to anyone who has taken an introductory programming class. Here it is given as an example of a BPEL process that accepts input, manipulates the input, and returns an output through a synchronous reply.

The input for a BPEL process comes from a client application. In this chapter, you will use Oracle BPEL Console as the client application. BPEL processes can also use a Java/JSP interface or a SOAP client as the client application. The client application has a partner link with the BPEL process. This link defines the role of each party, the types of data each accepts and returns, and so on. The WSDL file contains the definition of the partner links. Figure 3–1 provides an overview of how the Hello World process will work.

> **See Also:** `http://www.oracle.com/technology/bpel` for software download instructions and for details about running Oracle BPEL Process Manager and Eclipse BPEL Designer on Eclipse (including tutorials)

Figure 3–1 The Hello World BPEL Process



The Hello World process accepts the input from the partner link with a receive activity, which repackages the data into a variable. The greeting is then added using an assign activity, and the result is sent back through the partner link as output using the reply activity.

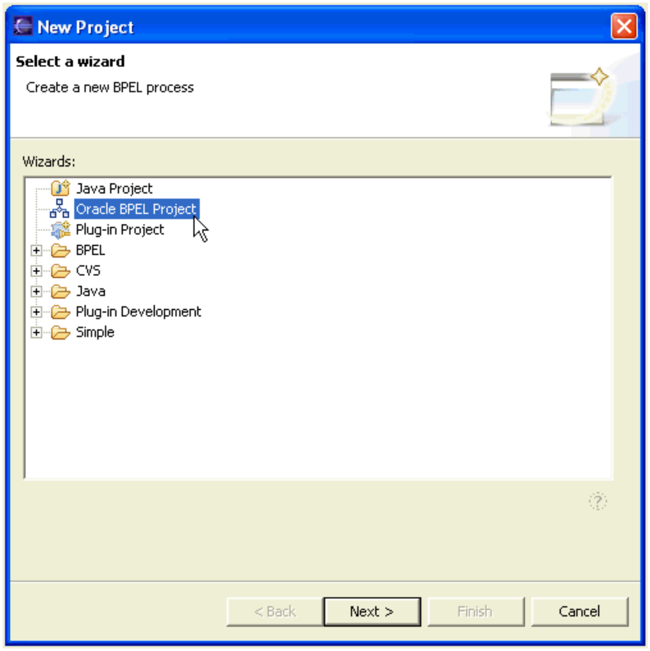
See Also: See "Reviewing the BPEL Source Code" section on page 3-9 for more information on these activities

Creating a New BPEL Project Using Eclipse BPEL Designer

You can use the New Project wizard in Eclipse BPEL Designer to generate automatically the skeleton of a BPEL project. This skeleton contains the BPEL source, a WSDL interface, a BPEL deployment descriptor, and an Ant script for compiling and deploying the BPEL process.

To create a new BPEL project:

1. Open Eclipse BPEL Designer by selecting **Start**, then **All Programs**, then **Oracle - *Oracle_Home***, then **Oracle BPEL Process Manager** ***version_number***, and then **BPEL Designer**.
2. Ensure that Oracle BPEL Server is running. See "Starting Oracle BPEL Process Manager Components" on page 2-2 for instructions.
3. From the **File** menu, click **New**, then **Project**.
4. Select **Oracle BPEL Project**.
5. Click **Next**.



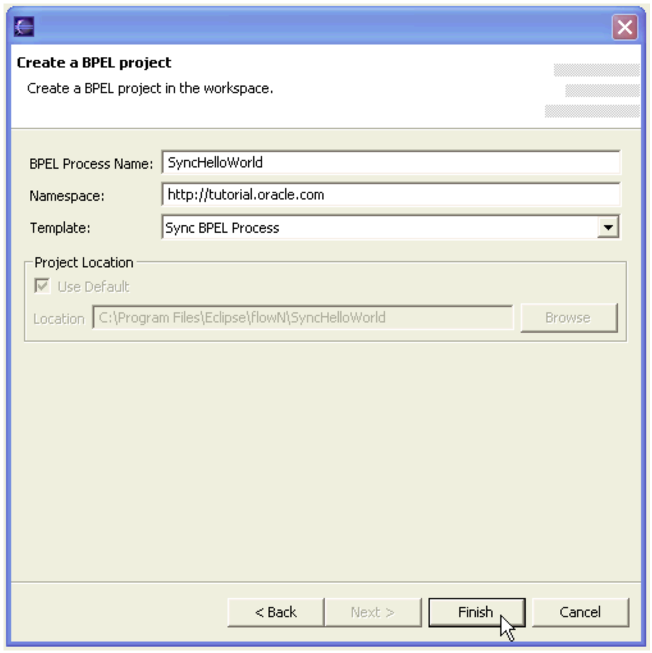
6. Enter **SyncHelloWorld** as the BPEL process name.

   Do not create a project names that begin with numbers.

7. (Optional) Enter **http://tutorial.oracle.com** as the namespace (replacing the one provided by default).

8. Change the template to **Synchronous BPEL Process**.

9. Leave the **Use default** check box selected, and click **Finish**.

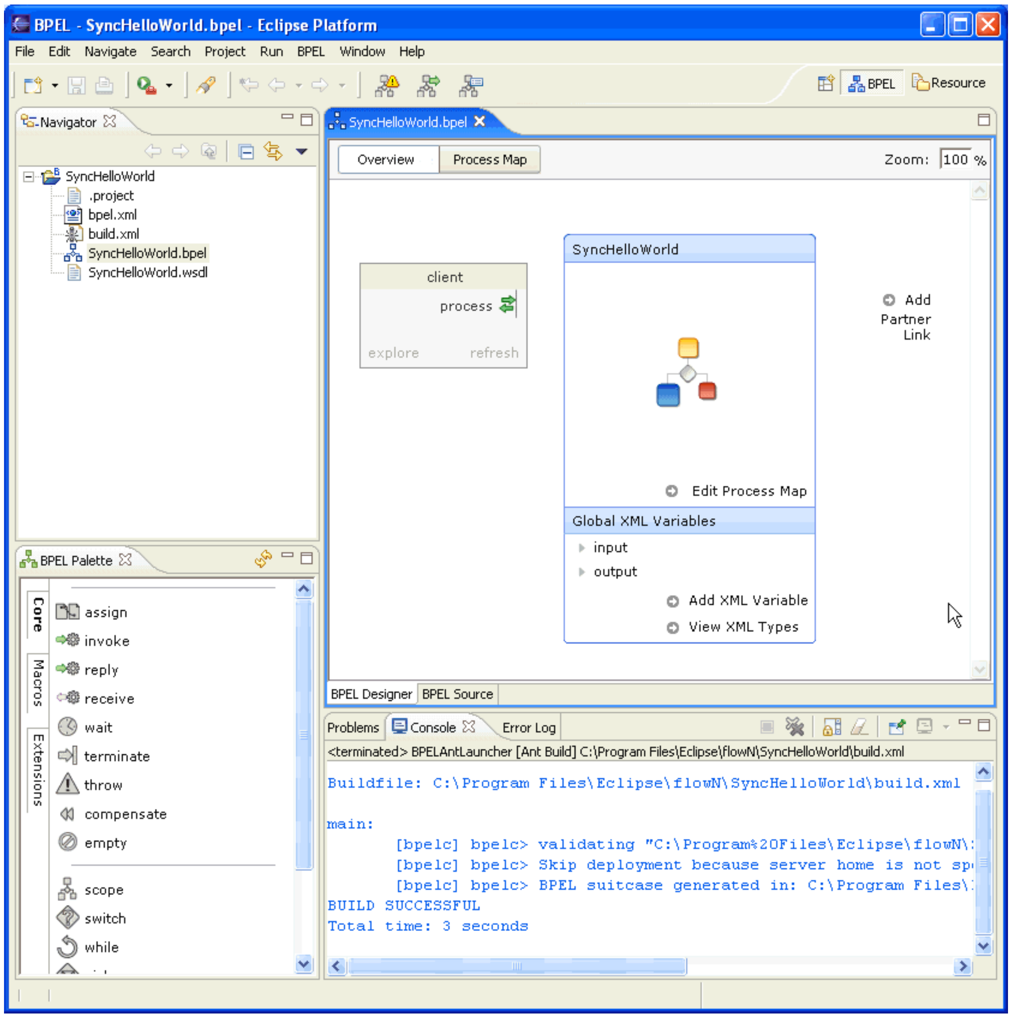

Create a BPEL project
Create a BPEL project in the workspace.

BPEL Process Name: SyncHelloWorld
Namespace: http://tutorial.oracle.com
Template: Sync BPEL Process
Project Location
[✓] Use Default
Location C:\Program Files\Eclipse\flowN\SyncHelloWorld

Browsing a New Project

The New Project wizard creates an initial set of project files for a new synchronous BPEL process, with all the necessary source files. Figure 3–2 provides an overview of how a new project appears in Eclipse BPEL Designer.

Figure 3–2 A New BPEL Process



```
<terminated> BPELAntLauncher [Ant Build] C:\Program Files\Eclipse\flowN\SyncHelloWorld\build.xml
Buildfile: C:\Program Files\Eclipse\flowN\SyncHelloWorld\build.xml

main:
       [bpelc] bpelc> validating "C:\Program%20Files\Eclipse\flowN\
       [bpelc] bpelc> Skip deployment because server home is not sp
       [bpelc] bpelc> BPEL suitcase generated in: C:\Program Files\
BUILD SUCCESSFUL
Total time: 3 seconds
```

Table 3–1 describes the initial project files that are created:

Table 3–1 BPEL Process Files

File	Description
.project	Eclipse BPEL Designer format project file.
bpel.xml	The deployment descriptor for the process. This file defines the locations of the WSDL files for services called by this process flow, along with other project-specific parameters.
build.xml	Apache Ant script for compiling and deploying this process.

Table 3–1 (Cont.) BPEL Process Files

File	Description
SyncHelloWorld.bpel	The BPEL source code for the process. The New Project wizard creates an empty flow, with just the minimum activities and definitions for the selected flow type. For a synchronous BPEL process, the only activities are a receive activity to initiate the flow from a synchronous client request and a reply activity to return the output.
SyncHelloWorld.wsdl	The WSDL (client) interface for this process. This file defines the input and output messages for this flow, the client interface and operations supported, and the BPEL `partnerLinkTypes`, so the flow can be incorporated into other processes. The New Project wizard generates a document-literal style WSDL that takes a string input message and returns a string response message.

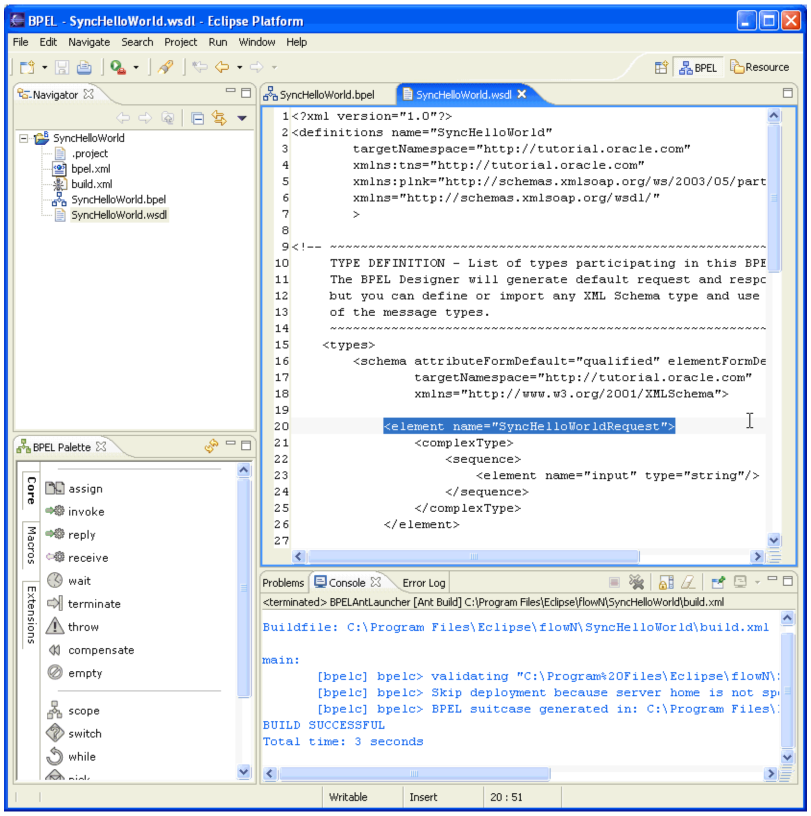
Viewing the WSDL Interface of a BPEL Process

After you create a new BPEL process, you can edit its WSDL file. WSDL files define the interface to the process flow, messages that it accepts and returns, operations that are supported, and so on. You can edit the WSDL file for the BPEL process in text mode in Eclipse BPEL Designer or in any other text editor. If you edit the WSDL file, you must refresh the client interface for the flow, if it is currently open in Eclipse BPEL Designer. In this tutorial, you will not edit the WSDL file; you only view it.

To view the input and output messages of the BPEL process:

1. In the Navigator, expand **SyncHelloWorld** and double-click the **SyncHelloWorld.wsdl** file.
2. Scroll through the WSDL file.

   The New Project wizard created the **SyncHelloWorldRequest complexType** element, which the flow accepts as input (in a document-literal style WSDL message), and a **SyncHelloWorldResponse** element that it returns.



```
<?xml version="1.0"?>
<definitions name="SyncHelloWorld"
        targetNamespace="http://tutorial.oracle.com"
        xmlns:tns="http://tutorial.oracle.com"
        xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/part
        xmlns="http://schemas.xmlsoap.org/wsdl/"
        >

<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
     TYPE DEFINITION - List of types participating in this BPE
     The BPEL Designer will generate default request and respc
     but you can define or import any XML Schema type and use
     of the message types.
     ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    <types>
        <schema attributeFormDefault="qualified" elementFormDe
                targetNamespace="http://tutorial.oracle.com"
                xmlns="http://www.w3.org/2001/XMLSchema">

            <element name="SyncHelloWorldRequest">
                <complexType>
                    <sequence>
                        <element name="input" type="string"/>
                    </sequence>
                </complexType>
            </element>

```

```
<terminated> BPELAntLauncher [Ant Build] C:\Program Files\Eclipse\flowN\SyncHelloWorld\build.xml
Buildfile: C:\Program Files\Eclipse\flowN\SyncHelloWorld\build.xml

main:
       [bpelc] bpelc> validating "C:\Program%20Files\Eclipse\flowN\:
       [bpelc] bpelc> Skip deployment because server home is not sp
       [bpelc] bpelc> BPEL suitcase generated in: C:\Program Files\
BUILD SUCCESSFUL
Total time: 3 seconds
```

Note: You can currently edit the WSDL file for the BPEL process in text mode in Eclipse BPEL Designer (or in any other text editor). If you edit the WSDL file (which you do not perform as part of this tutorial), you must refresh the client interface for the flow, if it is currently open in Eclipse BPEL Designer.

---

3. Close the (unmodified) **SyncHelloWorld.wsdl** window in the designer by clicking its **Close** button on the SyncHelloWorld.wsdl tab.

Switching Between the Overview, Process Map, and Source Code

This section describes how to view the BPEL file in three different ways.

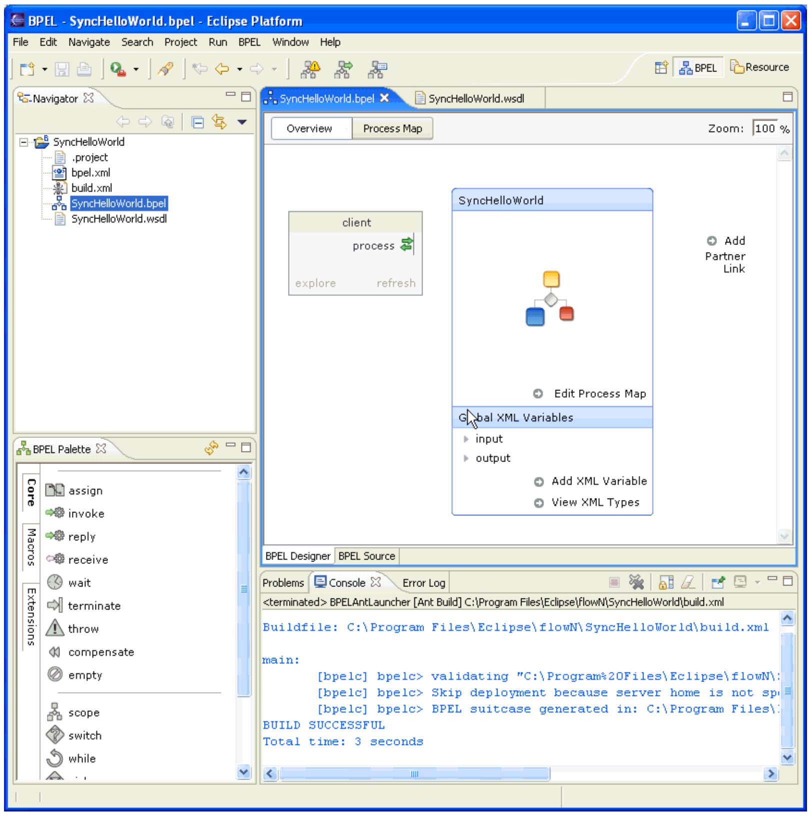
- Viewing an Overview of a BPEL Process
- Viewing a Detailed Process Map
- Viewing BPEL Source Code

Viewing an Overview of a BPEL Process

To display a visual overview of a BPEL process, double-click **SyncHelloWorld.bpel** in the **Navigator**, if it is not already open. Whenever you create a new BPEL project, the visual overview automatically opens.

On the left side of the overview is the client interface, which displays operations exposed by the process and any asynchronous callback operations (there are none for this process). Eclipse BPEL Designer automatically puts any partner link named **client** on the left side of the window. All other partner links are displayed in a list on the right.

Figure 3–3 Overview of the SyncHelloWorld Process



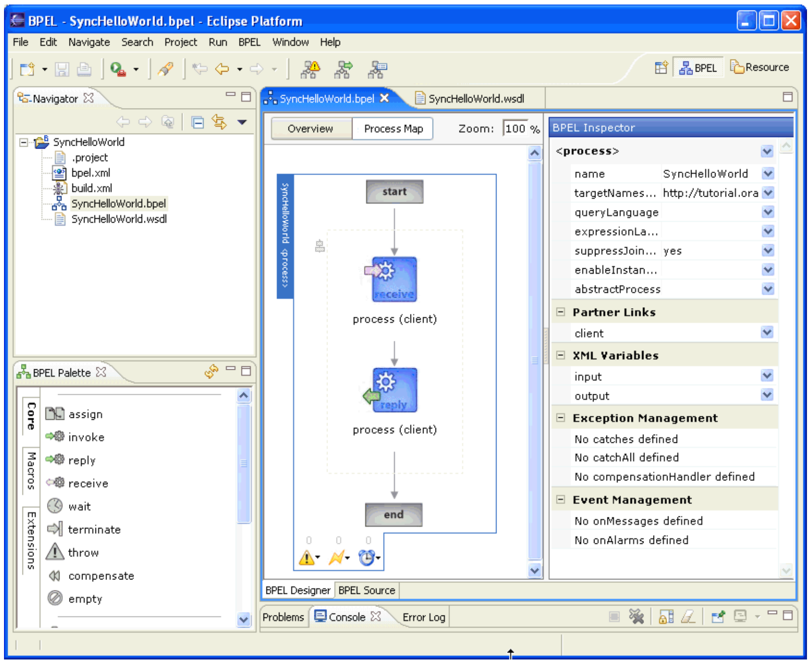
```
<terminated> BPELAntLauncher [Ant Build] C:\Program Files\Eclipse\flowN\SyncHelloWorld\build.xml
Buildfile: C:\Program Files\Eclipse\flowN\SyncHelloWorld\build.xml

main:
       [bpelc] bpelc> validating "C:\Program%20Files\Eclipse\flowN\:
       [bpelc] bpelc> Skip deployment because server home is not sp
       [bpelc] bpelc> BPEL suitcase generated in: C:\Program Files\
BUILD SUCCESSFUL
Total time: 3 seconds
```

Viewing a Detailed Process Map

The detailed process map provides a visual representation of the flow logic used for the process, from start to end. To display and edit the process map, click **Process Map** in the middle pane of the Eclipse BPEL Designer. Alternatively, you can click the **Edit Process Map** link. An **Inspector** pane appears on the right after you drill down into the **Process Map** view.

At the top of the window, the **Overview** and **Process Map** links allow you to shift from the current view (which is the **Overview** view) to the **Process Map** view, and back.

Figure 3–4 Detailed Process Map of the SyncHelloWorld Process



<process>

name	SyncHelloWorld
targetNames…	http://tutorial.ora
queryLanguage	
expressionLa…	
suppressJoin…	yes
enableInstan…	
abstractProcess	

Partner Links
client

XML Variables
input
output

Exception Management
No catches defined
No catchAll defined
No compensationHandler defined

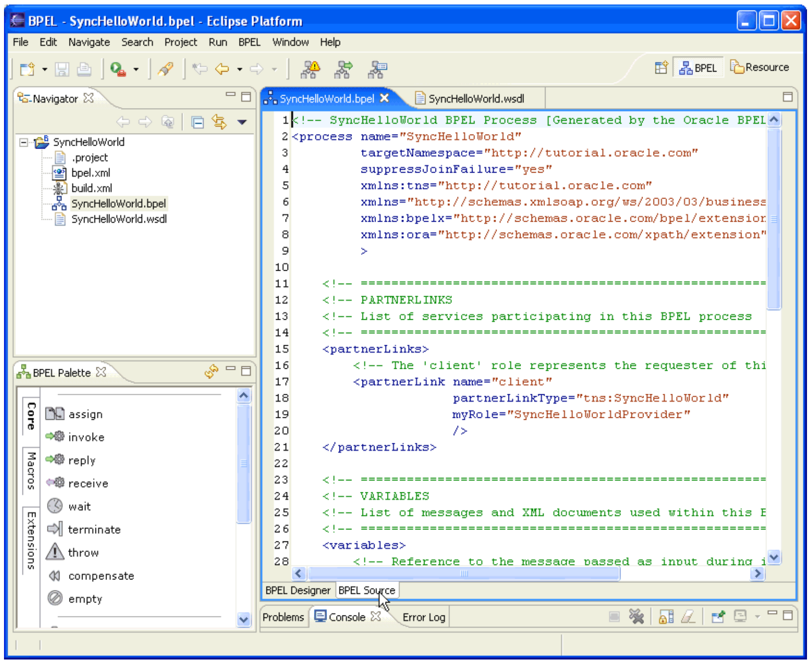
Event Management
No onMessages defined
No onAlarms defined

Viewing BPEL Source Code

At the bottom of the window, you can use the **BPEL Designer** and **BPEL Source** tabs to switch between graphical editing mode (**BPEL Designer**) and text-editing mode for the source code of the BPEL process (**BPEL Source**). Two-way editing is enabled; that is, changes you make in one mode are reflected in the other modes.

Click the **BPEL Source** tab of the editor window to display the syntax.

Figure 3–5 Source Code of SyncHelloWorld Process



```
<!-- SyncHelloWorld BPEL Process [Generated by the Oracle BPEL
<process name="SyncHelloWorld"
         targetNamespace="http://tutorial.oracle.com"
         suppressJoinFailure="yes"
         xmlns:tns="http://tutorial.oracle.com"
         xmlns="http://schemas.xmlsoap.org/ws/2003/03/business
         xmlns:bpelx="http://schemas.oracle.com/bpel/extension
         xmlns:ora="http://schemas.oracle.com/xpath/extension"
         >

    <!-- =====================================================
    <!-- PARTNERLINKS
    <!-- List of services participating in this BPEL process
    <!-- =====================================================
    <partnerLinks>
        <!-- The 'client' role represents the requester of thi
        <partnerLink name="client"
                     partnerLinkType="tns:SyncHelloWorld"
                     myRole="SyncHelloWorldProvider"
                     />
    </partnerLinks>

    <!-- =====================================================
    <!-- VARIABLES
    <!-- List of messages and XML documents used within this E
    <!-- =====================================================
    <variables>
        <!-- Reference to the message passed as input during i
```

Reviewing the BPEL Source Code

You can use **BPEL Source** mode to review the BPEL code generated by Eclipse BPEL Designer. This helps you to understand what the designer does, because everything you do in the designer is saved in standard BPEL source code. This also enables you to learn BPEL if you are not already familiar with it.

The source code has the following main components:

- Process Element
- Partner Links
- Global Variables
- Main Body of the Process

Process Element

The `process` element, which appears first, defines the name and target namespace you specified.

```
<process name="SyncHelloWorld"
     targetNamespace="http://tutorial.oracle.com"
     suppressJoinFailure="yes" xmlns:tns="http://tutorial.oracle.com"
     xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
     xmlns:bpelx="http://schemas.oracle.com/bpel/extension">
```

Partner Links

After the `process` element are the partner links, which define the other services or processes with which the process interacts. In this case, the only partner link created automatically by the wizard is the one for the client interface to this BPEL process.

```
<partnerLinks>
```

```
     <partnerLink name="client"
     <!-- comments... -->
          partnerLinkType="tns:SyncHelloWorld"
</partnerLinks>
          myRole="SyncHelloWorldProvider"/>
```

Global Variables

After the partner links is where you define any global variables that are accessible throughout the process. The New Project wizard automatically creates global variables for the `input` and `output` messages for the process. As you saw previously, the types for these two variables are defined in the WSDL for the process.

```
<variables>
     <!-- Reference to the message passed as input during initiation -->
     <variable name="input"
          messageType="tns:SyncHelloWorldRequestMessage"/>
     <!-- Reference to the message that will be returned to the
          requester
          -->
     <variable name="output"
          messageType="tns:SyncHelloWorldResponseMessage"/>
</variables>
```

Main Body of the Process

After the global variables is the main body of the process, which is any single BPEL activity. This is typically a compound activity that contains subactivities. When you create a new synchronous process, the New Project wizard creates a `sequence` (which in BPEL is a set of activities executed sequentially) that contains a `receive` activity to obtain the input message from the client to start the process. This is followed by a `reply` activity to return the result synchronously to the client.

```
<sequence name="main">
     <!-- Receive input from requester.
          Note: This maps to operation defined in SyncHelloWorld.wsdl
          -->
     <receive name="receiveInput" partnerLink="client"
          portType="tns:SyncHelloWorld"
          operation="process" variable="input"
          createInstance="yes"/>
     <reply name="replyOutput"
          partnerLink="client"
          portType="tns:SyncHelloWorld"
          operation="process"
          variable="output"/>
</sequence>
```

Finally, the `process` is closed:

```
</process>
```

In the next section, you can begin to edit this BPEL process to add logic to convert the input into a greeting.

Understanding the Sequence Activity

A sequence activity contains one or more activities that perform sequentially in the order in which they are listed within the `sequence` element; that is, in lexical order. The sequence activity completes when the final activity in the sequence has completed.

```
 <sequence standard-attributes>
    standard-elements
    activity+
  </sequence>
```

For example:

```
<sequence>
  <flow>
    ...
  </flow>
  <scope>
    ...
  </scope>
  <pick>
    ...
  </pick>
</sequence>
```

See Also: "Sequence Activity" on page C-14

Understanding the Assign Activity

The assign activity assigns values to variables. Copy rules are added to the assign to specify which data is being copied into which variable.

The following code sample from the `HelloWorld.bpel` file located at `C:\orabpel\samples\tutorials\101.HelloWorld` for Eclipse BPEL Designer.

```
<assign>
     <copy>
        <from expression="concat('Hello ',bpws:getVariableData('input',
          'payload','/tns:name'))"/>
        <to variable="output" part="payload" query="/result"/>
     </copy>
</assign>
```

If you are using JDeveloper BPEL Designer, this sample is available in *`Oracle_Home`*`\integration\orabpel\samples\tutorials\101.HelloWorld`.

Hello World is the first sample application used in the tutorials. It accepts input from the user, adds `Hello` with a trailing blank space to the beginning of the input payload, and copies the result to the output payload.

See Also:

- "Assign Activity" on page C-3
- `C:\orabpel\samples\references\Assign` (for Eclipse BPEL Designer)
- *`Oracle_Home`*`\integration\orabpel\samples\references\Assign` (for JDeveloper BPEL Designer)

Adding an Assign Activity to the Process Map

In this section, you learn how to add a new activity to the BPEL process. You will follow these steps:

1. Step 1: Viewing the Process Map
2. Step 2: Inserting an Assign Activity
3. Step 3: Adding a Copy Rule
4. Step 4: Defining the From Part (Source) of the Copy Rule
5. Step 5: Defining the To Part (Destination) of the Copy Rule

Step 1: Viewing the Process Map

Return to **BPEL Designer** mode and ensure that you are in **Process Map** view. You now see a graphical representation of the flow for the BPEL process. As described previously, the initial process created by the New Project wizard for a synchronous process just receives the input for the process operation from the client and returns the response with a synchronous reply.

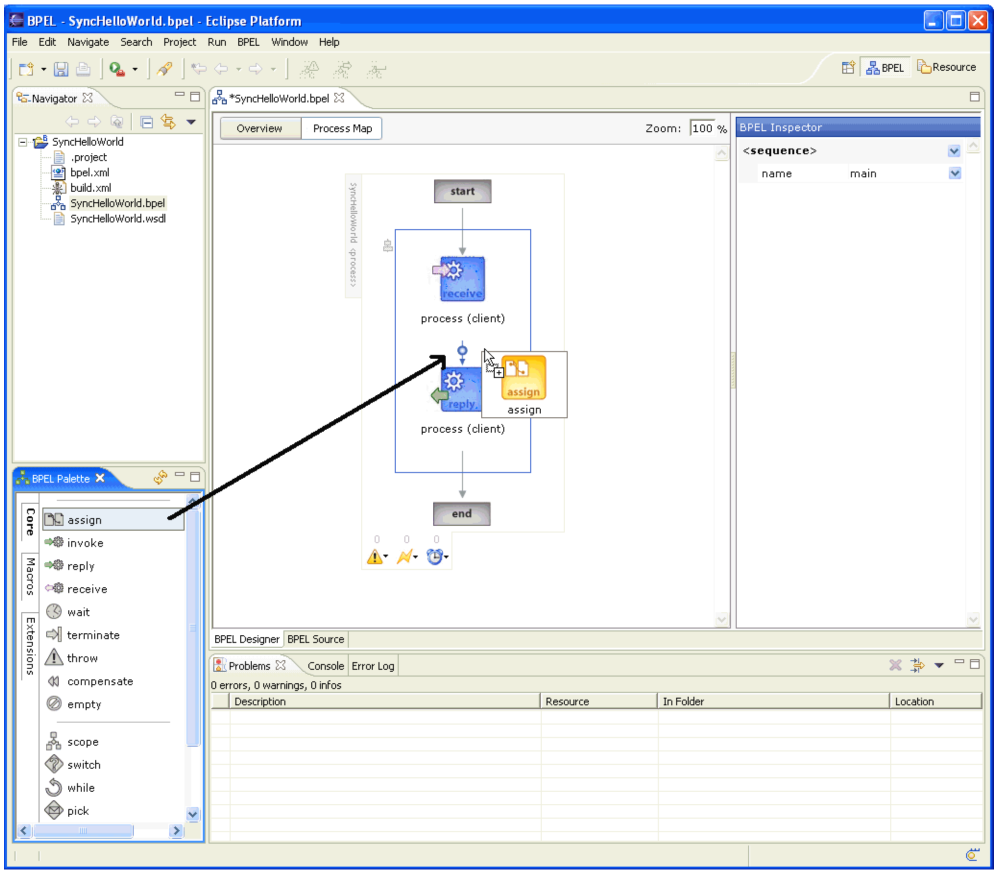
Step 2: Inserting an Assign Activity

You now insert an assign activity into the process. A BPEL assign activity provides a mechanism for doing simple data manipulation using XPath expressions.

You use the assign to concatenate the string `"Hello "` with the string name input message and copy the resulting string into the output message.

To insert a new activity into the process, find it in the **BPEL Palette** (you can click the **More Activities** link to see additional activities supported in **BPEL Designer** mode) and drag it to the location in the flow in which you want to insert it.

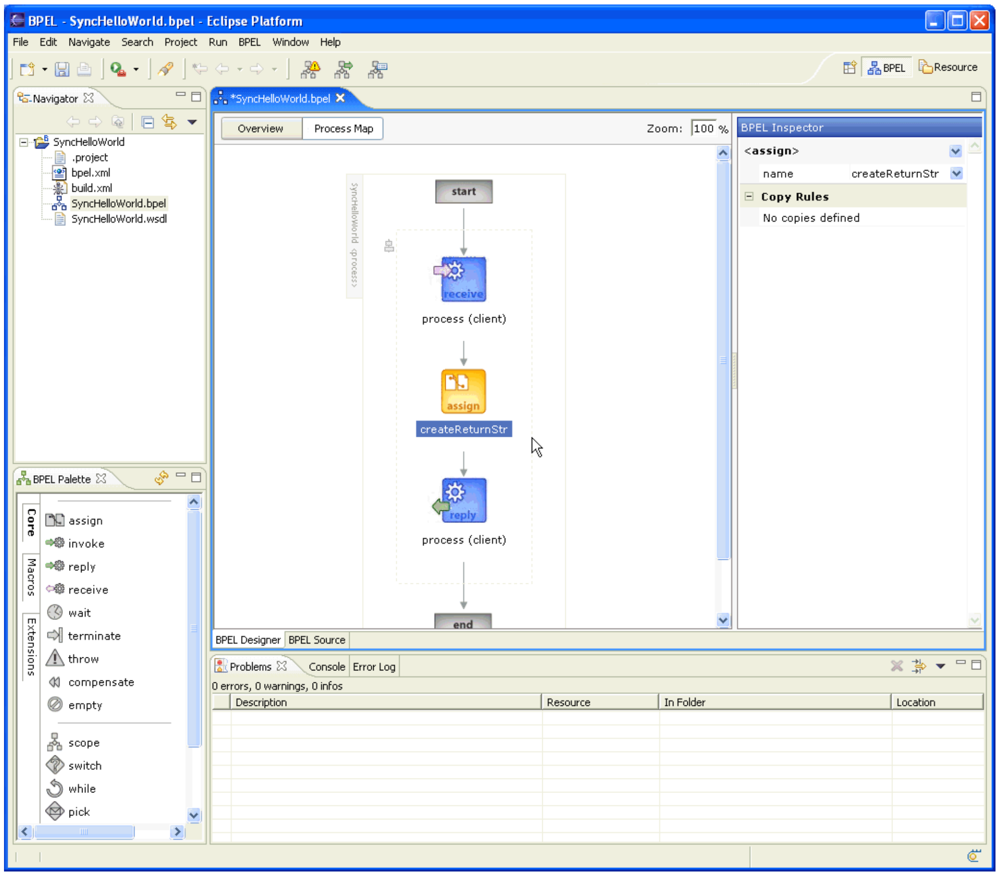
1. Drag an **assign** from the palette and drop it between the process **receive** activity and the **reply** response activity:



The new assign activity is initially selected. You can edit attributes for it in the **BPEL Inspector** window on the right.

2. Enter a value for the optional name attribute, such as **createReturnStr** (and press **Enter**).

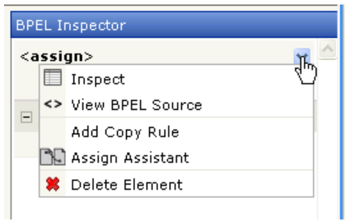
Note: The text field in which you enter the name is not selected until you click there.



Step 3: Adding a Copy Rule

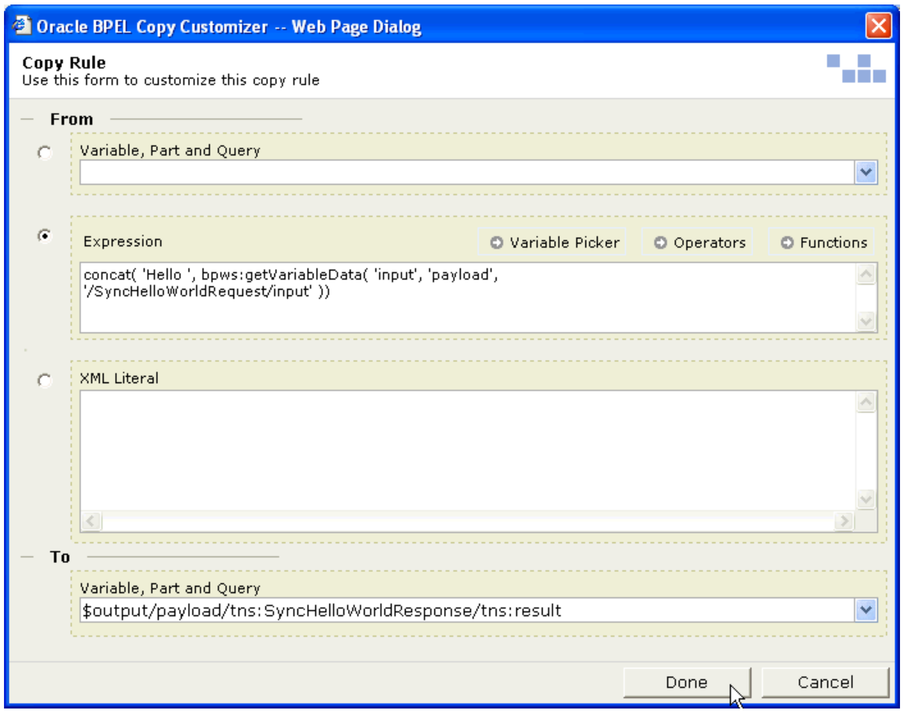
Next, you are ready to add a copy rule:

1. In the tasks drop-down list (the down arrow to the right of the **assign** activity heading), click **Add Copy Rule**.



This displays the Copy Rule window, which you use to perform the data manipulation. In BPEL, an assign statement can have many copy rules, each using variable data, XPath queries, XPath expressions, and literals to do simple data manipulation and transformation.

Each Copy Rule window for an assign activity has a **From** part, which specifies the source data, and a **To** part, which specifies a variable or element part as the destination for the data.



```
concat( 'Hello ', bpws:getVariableData( 'input', 'payload',
'/SyncHelloWorldRequest/input' ))
```

```
$output/payload/tns:SyncHelloWorldResponse/tns:result
```

Note: Oracle recommends that more complex data transformations be done using XQuery, XSLT, Java, commercial toolkits, or as a service. See `http://www.oracle.com/technology/bpel` for more information.

See Also: Chapter 14, "XSLT Mapper and Transformations"

Step 4: Defining the From Part (Source) of the Copy Rule

After you have added the copy rule, you are ready to define its From (source) part.

Click **Expression** in the **From** part. Enter the following expression in the text field:

```
concat( 'Hello ', bpws:getVariableData( 'input', 'payload',
 '/SyncHelloWorldRequest/input'))
```

Alternatively, use the XPath function wizard to complete this with the following steps:

1. Click the **...** context sensitive BPEL assistant button to start the BPEL Function Wizard.
2. Select the **contact** function and click **Next**.
3. Enter `"Hello "` for the firstString (note the trailing space after the **o**).
4. For the **nextString**, click the drop-down arrow to use the XPath picker and choose the Variable **input**, the Part **payload**, and the XPath Query **input**.
5. Click **Finish** and you see the same **concat** expression described previously (except that the wizard includes the name prefix **tns:** for the XPath expression).

Whether you enter it directly or use the wizard, this expression says to concatenate the literal string `Hello` with the string element that is accessed by the XPath query `/SyncHelloWorldRequest/input` within the part `payload` of the variable `input`.

See Also: The following documentation for more information on XPath expressions and queries:

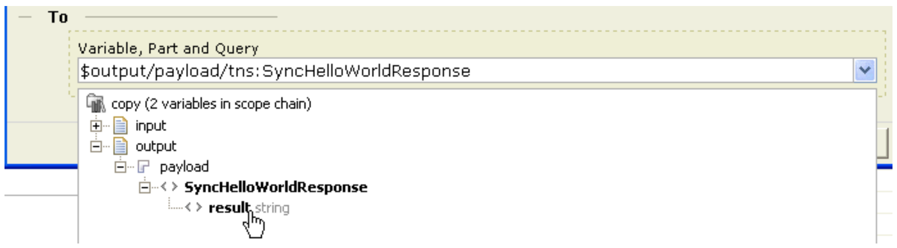
- The *XML Path Language (XPath) Specification* at http://www.w3.org/TR/1999/REC-xpath-19991116
- The *Business Process Execution Language for Web Services Specification* at http://www.ibm.com/developerworks/webservices/library/ws-bpel, which describes the `bpws:getVariableData` XPath function in Section 14.1
- The **BPEL Tutorials** link at http://www.oracle.com/technology/bpel, for tutorials 2 (Developing a Credit Flow BPEL Process) and 3 (Manipulating XML Documents in BPEL)

Step 5: Defining the To Part (Destination) of the Copy Rule

Next, you can complete the **To** section of the Copy Rule window to copy the concatenated string into the **helloString** element in the reply message.

Follow these steps:

1. In the **To** section, click the down arrow after the field.
2. Select **output**, then **payload**, then**SyncHelloWorldResponse**, and then **result string**:



$output/payload/tns:SyncHelloWorldResponse

This causes Eclipse BPEL Designer to automatically generate the XPath query string **$output/payload/tns:SyncHelloWorldResponse/tns:result** and enter it in the field.

Again, you can always enter queries directly into the field if you choose. Note that the wizard uses fully qualified names (including the namespaces) in the query string, but in both of these examples the namespaces are actually optional.

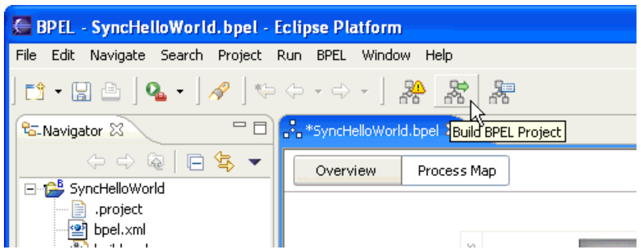
3. Click **Done**.

Next, you are ready to compile and deploy the BPEL process, and then test it.

Compiling and Deploying the BPEL Process

To compile and deploy the BPEL process:

1. Save the process.
2. Click the **Build BPEL Project** button on the **Workbench** toolbar to compile and deploy the process.


BPEL - SyncHelloWorld.bpel - Eclipse Platform
File Edit Navigate Search Project Run BPEL Window Help
Navigator
*SyncHelloWorld.bpel
Build BPEL Project
Overview
Process Map
SyncHelloWorld
.project
bpel.xml

This compiles the flow and packages all its components into a BPEL suitcase JAR file. You can then deploy this JAR file to Oracle BPEL Server by copying it into the appropriate deployment directory. The `build.xml` file generated by the wizard also automatically deploys the suitcase to the local server's default domain, in this case `C:\orabpel\domains\default\deploy`. (Note that this action is the same as running `obant` from the command line.)

Note: To see the contents of the BPEL suitcase JAR file, open the file `C:\orabpel\domains\default\deploy\bpel_SyncHelloWorld_1.0.jar` with any tool for opening JAR files (for example, WinZip).

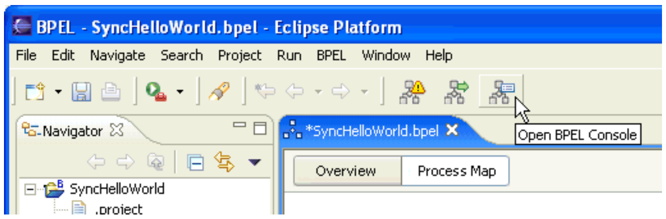
Testing the BPEL Process

After you have compiled and deployed the BPEL process, you can test it by using the automatically generated test interface in Oracle BPEL Console.

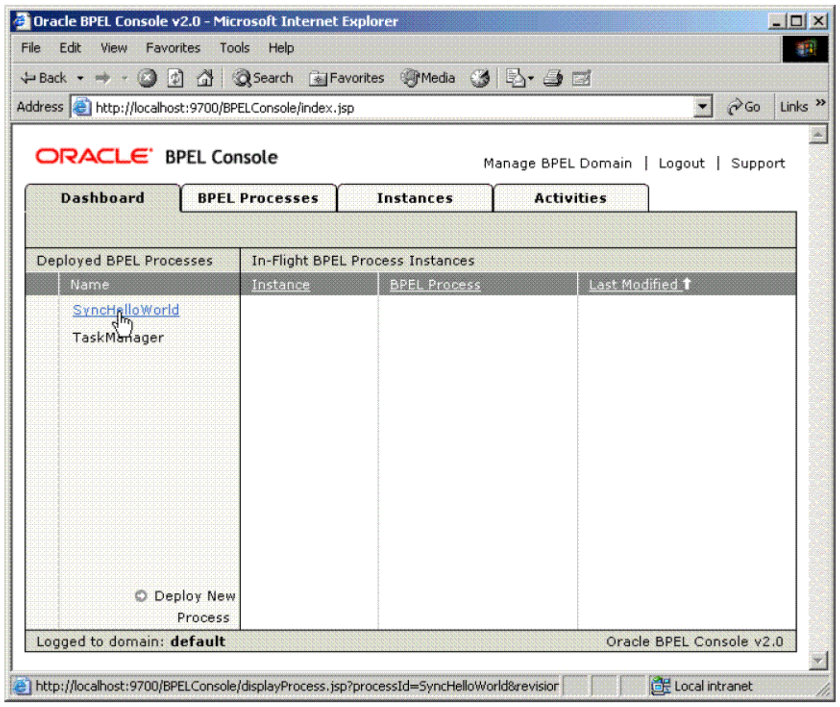
To test the SyncHelloWorld BPEL process:

1. Access the console at `http://localhost:9700/BPELConsole` or by selecting **Start**, then **All Programs**, then **Oracle -** ***Oracle_Home***, then **Oracle BPEL Process Manager** ***version_number***, and then **BPEL Console**.

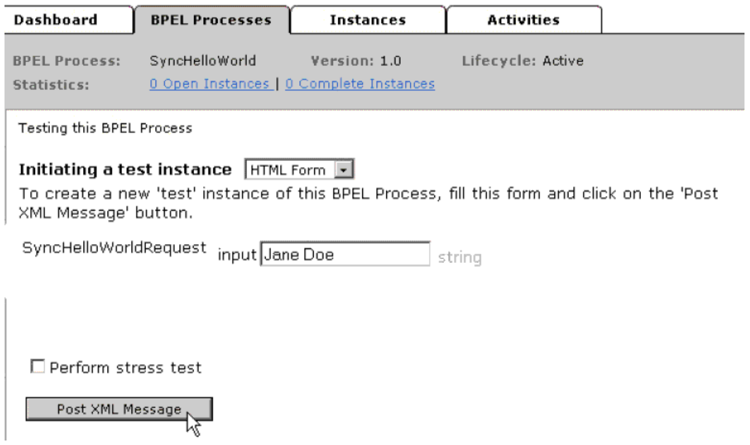
   Alternatively, you can click the **Open BPEL Console** button on the **Workbench** toolbar to open a browser window that points to the console.


BPEL - SyncHelloWorld.bpel - Eclipse Platform
File Edit Navigate Search Project Run BPEL Window Help
Navigator
*SyncHelloWorld.bpel
Open BPEL Console
Overview
Process Map
SyncHelloWorld
.project

2. Log in to the default domain to access the **Dashboard** tab:


Oracle BPEL Console v2.0 - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:9700/BPELConsole/index.jsp
Go Links
ORACLE BPEL Console
Manage BPEL Domain | Logout | Support
Dashboard
BPEL Processes
Instances
Activities
Deployed BPEL Processes
In-Flight BPEL Process Instances
Name
Instance
BPEL Process
Last Modified
SyncHelloWorld
TaskManager
Deploy New Process
Logged to domain: default
Oracle BPEL Console v2.0
http://localhost:9700/BPELConsole/displayProcess.jsp?processId=SyncHelloWorld&revisior
Local intranet

3. Click the **SyncHelloWorld** BPEL process link.
4. Complete the HTML test form interface.



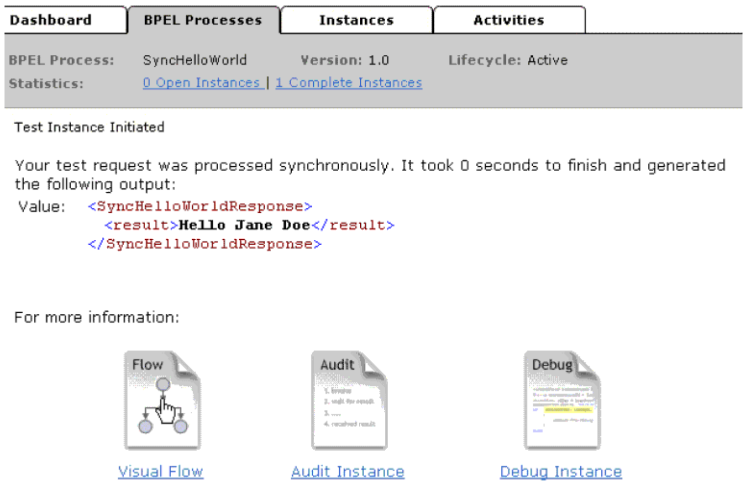
BPEL Process: SyncHelloWorld Version: 1.0 Lifecycle: Active
Statistics: 0 Open Instances | 0 Complete Instances

Testing this BPEL Process

To create a new 'test' instance of this BPEL Process, fill this form and click on the 'Post XML Message' button.

SyncHelloWorldRequest input Jane Doe string

5. Enter the name (for this example, **Jane Doe**) in the **input** field and click **Post XML Message** to initiate the process:



BPEL Process: SyncHelloWorld Version: 1.0 Lifecycle: Active
Statistics: 0 Open Instances | 1 Complete Instances

Test Instance Initiated

Your test request was processed synchronously. It took 0 seconds to finish and generated the following output:

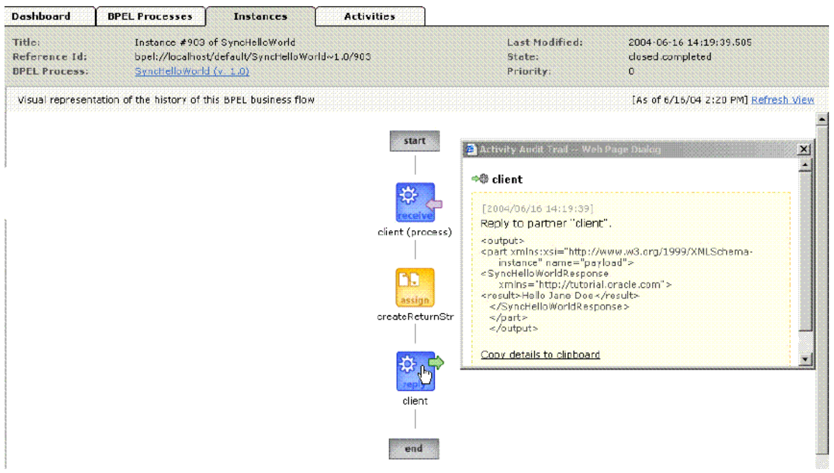
```
Value:   <SyncHelloWorldResponse>
           <result>Hello Jane Doe</result>
         </SyncHelloWorldResponse>
```

For more information:

You now see the results, because it is a synchronous operation.

6. To see the visual audit trail for this completed instance, click the **Visual Flow** link.

   The visual audit trail displays a representation of the current state and history of execution of the instance (which in this case has completed), with the same look and feel as Eclipse BPEL Designer. You can select activities in the audit trail to view their details. You can also click the **Audit** tab to see the text audit trail or the **Debug** tab to see the debugger for this instance.



Title:	Instance #903 of SyncHelloWorld	Last Modified:	2004-06-16 14:19:39.505
Reference Id:	bpel://localhost/default/SyncHelloWorld~1.0/903	State:	closed.completed
BPEL Process:	SyncHelloWorld (v. 1.0)	Priority:	0

Visual representation of the history of this BPEL business flow [As of 6/16/04 2:20 PM] Refresh View

```
[2004/06/16 14:19:39]
Reply to partner "client".
<output>
<part xmlns:xsi="http://www.w3.org/1999/XMLSchema-
    instance" name="payload">
<SyncHelloWorldResponse
    xmlns="http://tutorial.oracle.com">
<result>Hello Jane Doe</result>
  </SyncHelloWorldResponse>
  </part>
  </output>
```

7. Click the **reply** client activity at the bottom of the audit trail to see the returned XML message for the process, as shown previously.

Summary

This chapter discusses some of the basic concepts of a BPEL process, and shows the steps necessary to create a BPEL process. These steps are shown in Eclipse BPEL Designer, but you can also use JDeveloper BPEL Designer. This process is run from Oracle BPEL Console, and is a synchronous Web service that can be used as a building block for other BPEL processes or invoked using standard Web services interfaces.

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 4, "Manipulating XML Data in BPEL"
- Chapter 5, "Invoking a Synchronous Web Service"
- Chapter 6, "Calling an Asynchronous Web Service"
- Chapter 7, "Parallel Flow"
- Chapter 8, "Conditional Branching"
- Chapter 9, "Fault Handling"
- Chapter 10, "Incorporating Java/J2EE Code in BPEL Processes"
- Chapter 11, "Events and Timeouts"
- Chapter 12, "Invoking a BPEL Process"
- Chapter 13, "Interaction Patterns"

4 Manipulating XML Data in BPEL

This chapter describes how to manipulate XML data in BPEL, including the large role that XPath expressions play in manipulating XML data.

This chapter contains the following topics:

- How XML Data Works in BPEL
- About Data Manipulation and XPath Standards
- Initializing a Variable with Expression Constants or Literal XML
- Copying Between Variables
- Accessing Fields within Complex Type 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 Sequences/Arrays
- Converting from a String to an XML Element
- Differences Between Document-Style and RPC-Style WSDL Files
- Summary

See Also: The sample files located at `C:\orabpel\samples\tutorials\103.XMLDocuments` for Eclipse BPEL Designer and `*Oracle_Home*\integration\orabpel\samples\tutorials\103.XMLDocuments` for JDeveloper BPEL Designer

---

Note: Most 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 4-16.

How XML Data Works in BPEL

In a BPEL process, every piece of data is XML. This includes the messages passed to and from the BPEL process, the messages exchanged with external services, and local variables used by the flow. You define the types for these messages and variables with the XML schema, usually in the WSDL file for the flow or in the WSDL files for the services it invokes. 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 also available that involve using XQuery, XSLT, or Java (usually to do more complex data transformation or manipulation). This chapter reviews the various methods from the simplest to the most advanced. The explanations are largely by example, providing an introduction to the supporting specifications without repeating their details. The rest of 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 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* specification is as follows:

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

This syntax is described in detail in that specification. The `from-spec` and `to-spec` typically specify a variable or variable part, as in:

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

Rather than repeating all syntax details, this chapter shows and describes excerpts taken primarily from sample projects provided in the `C:\orabpel\samples\references` directory for Eclipse BPEL Designer and *`Oracle_Home`*`\integration\orabpel\references` directory for JDeveloper BPEL Designer.

Note: If you used the tutorials with either BPEL designer (JDeveloper BPEL Designer or Eclipse BPEL Designer) to add an assign activity to the process, you supplied details about the activity in a Copy Rule window that included a **From** section and a **To** section. This reflects the underlying BPEL source code syntax shown previously in this section.

XPath standards play a key role in the assign activity, as summarized in this section. 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 is used to select 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, as shown in the following example:

  ```
  <from variable="input" part="payload"
        query="/p:CreditFlowRequest/p:ssn">
  ```

  For XPath version 1.0, the value of the query attribute must be an absolute 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:

  ```
  <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 includes support for a large number of built-in functions, including functions for string manipulation (such as `concat`), numeric functions (like `sum`), and others.

  ```
  <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 `http://schemas.xmlsoap.org/ws/2003/03/business-process/` and indicated by the prefix `bpws`:

  ```
  <from expression= "bpws:getVariableData('input', 'payload', '/p:value') + 1"/>
  ```

  For more information, see sections 9.1 and 14.1 of the *Business Process Execution Language for Web Services Specification*.

- **Oracle BPEL Extension XPath 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:`. If you have Oracle BPEL Process Manager installed locally, Oracle BPEL Console lists and describes all available XPath functions through the following URL: `http://localhost:9700/BPELConsole/domain.jsp?mode=xpath`.

- **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 `C:\orabpel\domains\default\config\xpath-functions.xml` file for Eclipse BPEL Designer or the *`Oracle_Home`*`\integration\orabpel\domains\default\config\xpath-functions.xml` file for JDeveloper BPEL Designer.

  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 or Java, 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:
>
> - `C:\orabpel\samples\tutorials\114.XSLTTransformations` (for Eclipse BPEL Designer)
> - *`Oracle_Home`*`\integration\orabpel\samples\tutorials\114.XSLTTransformations` (for JDeveloper BPEL Designer)
> - Chapter 14, "XSLT Mapper and Transformations"

The following sections show related definitions in the BPEL and WSDL files that help explain the examples.

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.

This example assigns a literal `result` element to the `payload` part of the `output` variable.

```
<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:

- `C:\orabpel\samples\references\Assign` (for Eclipse BPEL Designer)
- *`Oracle_Home`*`\integration\orabpel\samples\references\Assign` (for JDeveloper BPEL Designer)

Copying Between Variables

Although this type of manipulation is not very commonly done, it does serve as a good starting point for illustrating copying between variables or variable parts, because of its simplicity. It copies 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.

```
<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:

```
<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:

```
<message name="person" xmlns:x="http://tempuri.org/bpws/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 Complex Type Variables

Given the types of definitions present in most WSDL files, you must go down to the level of copying from or to a field within part of a complex type variable. 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.

```
<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:

```
<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. It message type, `CreditFlowRequestMessage`, is defined in `CreditFlowService.wsdl` as follows:

```
<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:

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

See Also: The following samples:

- `C:\bpelz\workspace\CreditFlow` (for Eclipse BPEL Designer)
- `C:\orabpel\samples\utils\CreditRatingService` (for Eclipse BPEL Designer)
- *`Oracle_Home`*`\integration\orabpel\samples\utils\CreditRatingService` (for JDeveloper BPEL Designer)

Assigning Numeric Values

The following example shows how to assign an XPath expression with the integer value `100`.

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

```
</assign>
```

See Also: The following samples:

- `C:\orabpel\samples\references\Assign` (for Eclipse BPEL Designer)
- *Oracle_ Home*`\integration\orabpel\samples\references\Assign` (for JDeveloper BPEL Designer)

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).

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

See Also: The following samples:

- `C:\orabpel\samples\references\Assign` (for Eclipse BPEL Designer)
- *Oracle_ Home*`\integration\orabpel\samples\references\Assign` (for JDeveloper BPEL Designer)

Assigning String Literals

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

```
<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 samples:

- `C:\orabpel\samples\references\Assign` (for Eclipse BPEL Designer)
- *Oracle_ Home*`\integration\orabpel\samples\references\Assign` (for JDeveloper BPEL Designer)

Concatenating Strings

Rather than copy the value of one string variable (or variable part or field) to another, you may first want to perform string manipulation, such as concatenating several strings together. In Chapter 3, "Building a Simple BPEL Process", the string `'Hello '` was concatenated with a name supplied in an input message. A similar example from the assign reference sample is shown in the following syntax. The concatenation is accomplished with the core XPath function named `concat`; in addition, the variable 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.

```
<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*.

See Also: The following samples:

- `C:\orabpel\samples\references\Assign` (for Eclipse BPEL Designer)
- *`Oracle_Home`*`\integration\orabpel\samples\references\Assign` (for JDeveloper BPEL Designer)

Assigning Boolean Values

In this example, 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.

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

See Also: The following samples:

- `C:\orabpel\samples\references\Assign` (for Eclipse BPEL Designer)
- *`Oracle_Home`*`\integration\orabpel\samples\references\Assign` (for JDeveloper BPEL Designer)

Assigning Date or Time

You can assign the current value of a date or time field by using the 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 BPEL XPath function `formatDate`.

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, 2003'` (and assigns it to the string field `formattedDate`).

```
<!-- execute the XPath extension function formatDate() -->
<assign>
   <copy>
      <from expression="ora:formatDate('2003-06-10T15:56:00',
         'MMM dd, yyyy')"/>
      <to variable="output" part="payload"
         query="/invoice/formattedDate"/>
   </copy>
</assign>
```

See Also: The following samples:

- `C:\orabpel\samples\references\XPathFunction` (for Eclipse BPEL Designer)
- *`Oracle_Home`*`\integration\orabpel\samples\references\XPathFunction` (for JDeveloper BPEL Designer)

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`:

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

The namespace prefixes in this example are optional and not integral to the example.

The WSDL file defines a customer to have a type in which `custId` is defined as an attribute, as follows:

```
<complexType name="CustomerProfileType">
   <sequence>
      <element name="name" type="string"/>
      ...
   </sequence>
   <attribute name="custId" type="string"/>
</complexType>
```

See Also: The following samples:

- `C:\orabpel\samples\references\XPath` (for Eclipse BPEL Designer)
- *`Oracle_Home`*`\integration\orabpel\samples\references\XPath` (for JDeveloper BPEL Designer)

Manipulating XML Data Sequences/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 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 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`.

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:

- `C:\orabpel\samples\tutorials\112.Arrays` (for Eclipse BPEL Designer)
- *`Oracle_ Home`*`\integration\orabpel\samples\tutorials\112.Arrays` (for JDeveloper BPEL Designer)

Statically Indexing into an XML Data Sequence

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:

```
<assign>
   <!-- get the first address and assign to variable address -->
   <copy>
      <from variable="input" part="payload"
         query="/tns:invalidLoanApplication/autoloan:application
                /autoloan:customer/autoloan:addresses[1]"/>
      <to variable="address"/>
   </copy>
</assign>
```

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`).

```
<assign>
   <!-- get the first address's street and assign to street1 -->
   <copy>
      <from variable="input" part="payload"
         query="/tns:invalidLoanApplication/autoloan:application
               /autoloan:customer/autoloan:addresses[position()=1]
               /autoloan:street"/>
      <to variable="street1"/>
   </copy>
</assign>
```

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 you prefer.

See Also: The following samples:

- `C:\orabpel\samples\references\XPath` (for Eclipse BPEL Designer)
- *`Oracle_ Home`*`\integration\orabpel\samples\references\XPath` (for JDeveloper BPEL Designer)

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`:

```
<assign>
   <copy>
      <from expression="count(bpws:getVariableData('outpoint', 'payload',
                        '/p:invoice/p:lineItems/p:item')"/>
      <to variable="lineItemSize"/>
   </copy>
</assign>
```

See Also: The following samples:

- `C:\orabpel\samples\references\XPathFunction` (for Eclipse BPEL Designer)
- *`Oracle_Home`*`\integration\orabpel\samples\references\XPathFunction` (for JDeveloper BPEL Designer)

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:

- Dynamic Indexing Example
- Appending New Items to a Sequence
- Merging Data Sequences
- Dynamically Indexing with the BPEL getElement Function
- Merging Data Sequences/Arrays
- Appending New Items to a Sequence/Array

Dynamic Indexing Example

The dynamic indexing method shown here applies a trailing XPath to the result of `bwps: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, `[...]`):

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

```
<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 `bwps: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:

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

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

For example:

```
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.

Appending New Items to a Sequence

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

```
<assign name="assign-3">
    <copy>
        <from expression="bpws:getVariableData('idx')+1" />
        <to variable="idx"/>
    </copy>
    <bpelx:append>
        <bpelx:from variable="partInfoResultVar" part="payload" />
        <bpelx:to variable="output" part="payload" />
    </bpelx:append>
    ...
</assign>
```

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 order words, the payload element of `output` variable is used as the parent element.

See Also: The following samples:

- `C:\orabpel\samples\tutorials\126.DataAggregator\AggregationTutorial` (for Eclipse BPEL Designer)
- *`Oracle_Home`*`\integration\orabpel\samples\tutorials\126.DataAggregator\AggregationTutorial` (for JDeveloper BPEL Designer)

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).

To accomplish the merging logic, use the following two `append` operations under `assign`:

```
<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:

- `C:\orabpel\samples\tutorials\112.Arrays` (for Eclipse BPEL Designer)
- *`Oracle_Home`*`\integration\orabpel\samples\tutorials\112.Arrays` (for JDeveloper BPEL Designer)

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` 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.

```
<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>
```

See Also: The following samples:

- `C:\orabpel\samples\references\XPathFunction` (for Eclipse BPEL Designer)
- *`Oracle_ Home`*`\integration\orabpel\samples\references\XPath Function` (for JDeveloper BPEL Designer)

Merging Data Sequences/Arrays

You can merge two sequences of compatible types into a single sequence. To do so, use the XPath function `mergeChildNodes`.

```
<!-- execute the XPath extension function mergeChildNodes(e1, e2) and assign to a
variable -->
<assign>
   <copy>
      <from expression="ora:mergeChildNodes(
         bpws:getVariableData('input', 'payload', '/invoice/lineItems'),
         bpws:getVariableData('literalLineItems'))"/>
      <to variable="mergedLineItems"/>
   </copy>
</assign>
```

See Also: The following samples:

- `C:\orabpel\samples\references\XPathFunction` (for Eclipse BPEL Designer)
- *`Oracle_ Home`*`\integration\orabpel\samples\references\XPath Function` (for JDeveloper BPEL Designer)

Appending New Items to a Sequence/Array

Use the BPEL XPath function `addChildNode` enables BPEL processes to append new elements to an existing sequence.

```
<!-- execute the XPath extension function addChildNode(Element e, Node childNode)
-->
<assign>
   <copy>
      <from expression="ora:addChildNode(bpws:getVariableData('output',
         'payload', '/invoice/lineItems'),
          bpws:getVariableData('escapedLineItem'))"/>
      <to variable="output" part="payload" query="/invoice/lineItems"/>
   </copy>
</assign>
```

See Also: The following samples:

- `C:\orabpel\samples\references\XPathFunction` (for Eclipse BPEL Designer)
- *`Oracle_ Home`*`\integration\orabpel\samples\references\XPath Function` (for JDeveloper BPEL Designer)

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 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.

```
<!-- execute the XPath extension function
parseEscapedXML('&lt;item&gt;') and assign to a variable
-->
<assign>
   <copy>
      <from expression="ora:parseEscapedXML(
         '&lt;item xmlns=&quot;http://samples.otn.com&quot;
                   sku=&quot;006&quot;&gt;
          &lt;description&gt;sun ultra sparc VI server
          &lt;/description&gt;
          &lt;price&gt;1000
          &lt;/price&gt;
          &lt;quantity&gt;2
          &lt;/quantity&gt;
          &lt;lineTotal&gt;2000
          &lt;/lineTotal&gt;
          &lt;/item&gt;')"/>
      <to variable="escapedLineItem"/>
   </copy>
</assign>
```

See Also: The following samples:

- `C:\orabpel\samples\references\XPathFunction` (for Eclipse BPEL Designer)
- *`Oracle_Home`*`\integration\orabpel\samples\references\XPathFunction` (for JDeveloper BPEL Designer)

Differences Between Document-Style and RPC-Style WSDL Files

All of 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:

```
<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:

```
<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

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

<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>
```

BPEL

```
<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 samples:

- `C:\orabpel\samples\utils\AsyncLoanService` (`LoanServices` for Eclipse BPEL Designer)
- `C:\orabpel\samples\demos\LoanDemo\LoanFlow` (BPEL demo application for Eclipse BPEL Designer)
- *Oracle_ Home*`\integration\orabpel\samples\utils\AsyncLoanService` (`LoanServices` for JDeveloper BPEL Designer)
- *Oracle_ Home*`\integration\orabpel\samples\demos\LoanDemo\LoanFlow` (BPEL demo application for JDeveloper BPEL Designer)

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.

5

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 receive the reply using a synchronous callback. This chapter explains how to establish a partner link and set up a synchronous callback. Only one port type is required for a synchronous callback.

This chapter discusses the components necessary to perform a synchronous callback, examines how these components are coded, and shows how to set up a synchronous callback using Eclipse BPEL Designer.

This chapter contains the following topics:

- Use Case
- Synchronous Service Concepts
- Calling a Synchronous Service
- Summary

Use Case

This chapter uses an example of a BPEL process sending a stock code to a Web service and receiving a stock quote in return.

See Also: The following sample files:

- `C:\orabpel\samples\tutorials\104.SyncQuoteConsumer`
- *`Oracle_ Home`*`\integration\orabpel\samples\tutorials\104.SyncQuoteConsumer`

Synchronous Service Concepts

The key concepts necessary for a synchronous callback are the partner link and the invoke activity. A 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.

The 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 60 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.

Examples

This section examines how synchronous functionality is defined in the stock quote Web service's `StockQuoteService.wsdl` file (the Web service to be called) and the client's `QuoteConsumer.bpel` file and `bpel.xml` deployment description file.

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

If using Eclipse BPEL Designer:

- `C:\orabpel\samples\tutorials\104.SyncQuoteConsumer\QuoteConsumer.bpel`
- `C:\orabpel\samples\tutorials\104.SyncQuoteConsumer\bpel.xml`
- `C:\orabpel\samples\tutorials\104.SyncQuoteConsumer\QuoteConsumer.wsdl`
- `C:\orabpel\samples\utils\StockQuoteService\StockQuoteService.wsdl`

If using JDeveloper BPEL Designer:

- *Oracle_Home*`\integration\orabpel\samples\tutorials\104.SyncQuoteConsumer.bpel`
- *Oracle_Home*`\integration\orabpel\samples\tutorials\104.SyncQuoteConsumer\bpel.xml`
- *Oracle_Home*`\integration\orabpel\samples\tutorials\104.SyncQuoteConsumer.wsdl`
- *Oracle_Home*`\integration\orabpel\samples\utils\104.StockQuoteService.wsdl`

The Partner Link

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 partner link definition is as follows:

```
<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>
```

Partner links are followed by global variable definitions that are accessible throughout the BPEL process. The types for these variables are defined in the WSDL for the process itself.

```
<variables>
   <!-- Reference to the message passed as input during initiation -->
   <variable name="input" messageType="tns:QuoteConsumerRequestMessage"/>
   <!-- Reference to the message that will be sent back to the
```

```
            requestor during callback
            -->
   <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.

Port Types

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.

partnerLinkTypes for Synchronous Services

This section examines the Web service's `.wsdl` file, and identifies the sections of the file that enable it to work with BPEL processes.

View the `partnerLinkType` section of the `QuoteConsumer.wsdl` file. The `partnerLinkType` 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

```
<!--
  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>
```

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 -->
    <portType name="QuoteConsumer">
        <operation name="initiate">
            <input message="tns:QuoteConsumerRequestMessage"/>
        </operation>
```

```
</portType>

<!-- portType implemented by the requester of QuoteConsumer BPEL process
     for asynchronous callback purposes
     -->
<portType name="QuoteConsumerCallback">
    <operation name="onResult">
        <input message="tns:QuoteConsumerResultMessage"/>
    </operation>
</portType>
```

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.

UDDI and WSIL Directories

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 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 the Web Services Description Language (WSDL).

The Invoke Activity

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

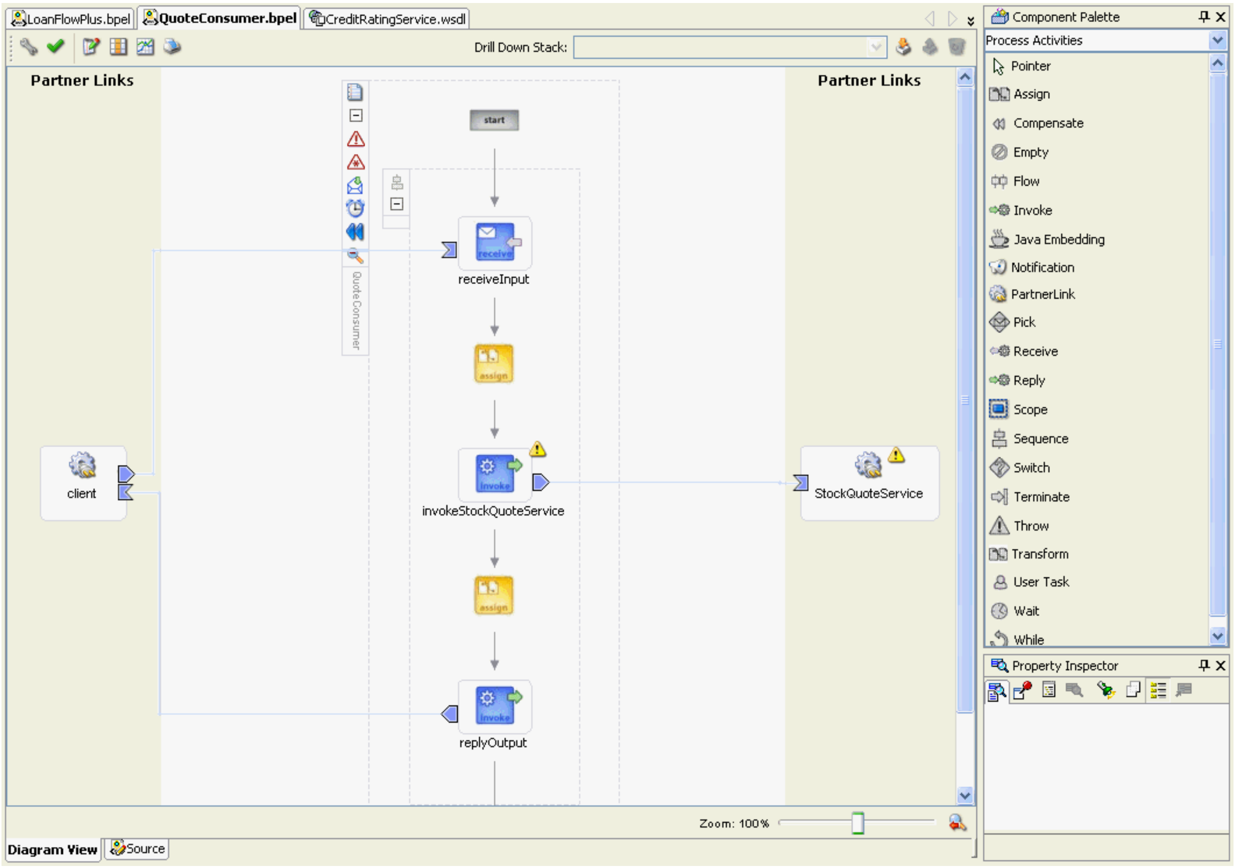
```
<sequence>
<!-- Receive input from requestor.
             Note: This maps to operation defined in QuoteConsumer.wsdl
             -->
<receive name="receiveInput" partnerLink="client" portType="samples:QuoteConsumer"
operation="initiate" variable="input" createInstance="yes"/>
<assign>
<copy>
<from 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>
```

Calling a Synchronous Service

This section examines the **QuoteConsumer.bpel** application to explore the concepts involved with a synchronous callback operation. You may import the project file at this location to examine the application. For a more step-by-step approach, see `http://www.oracle.com/technology/bpel` and download the files under **BPEL Training Materials**.

An overview of the **QuoteConsumer.bpel** file in JDeveloper BPEL Designer reveals a simple application with five activities:



The **receive** activity receives input from the user, as defined in the **QuoteConsumer.wsdl** file. The first **assign** activity packages the data from the client in a way that is accepted by the **QuoteConsumer** service. The synchronous callback labeled **process** then sends the repackaged data to the **QuoteConsumer** service and receives a response. This response is repackaged by a second assign activity into a form that can be accepted by the client application. Finally, the repackaged response is sent back to the client.

The following BPEL code that performs the synchronous callback is on lines 61-91:

```
     <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"/>



<!-- Generate content of output message based on the content of the
             input message.
             -->
     <assign>
        <copy>
           <from variable="response" part="result" query="/result"/>
           <to variable="output" part="payload" query="/tns:result"/>
         </copy>
     </assign>
```

See Also: The following samples:

- `C:\orabpel\samples\tutorials\104.SyncQuoteConsumer` (for Eclipse BPEL Designer)
- *`Oracle_Home`*`\integration\orabpel\samples\tutorials\104.SyncQuoteConsumer` (for JDeveloper BPEL Designer)

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.

6 Calling an Asynchronous Web Service

This chapter describes how to call an asynchronous Web service. Asynchronous messaging styles are extremely 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.

This chapter contains the following topics:

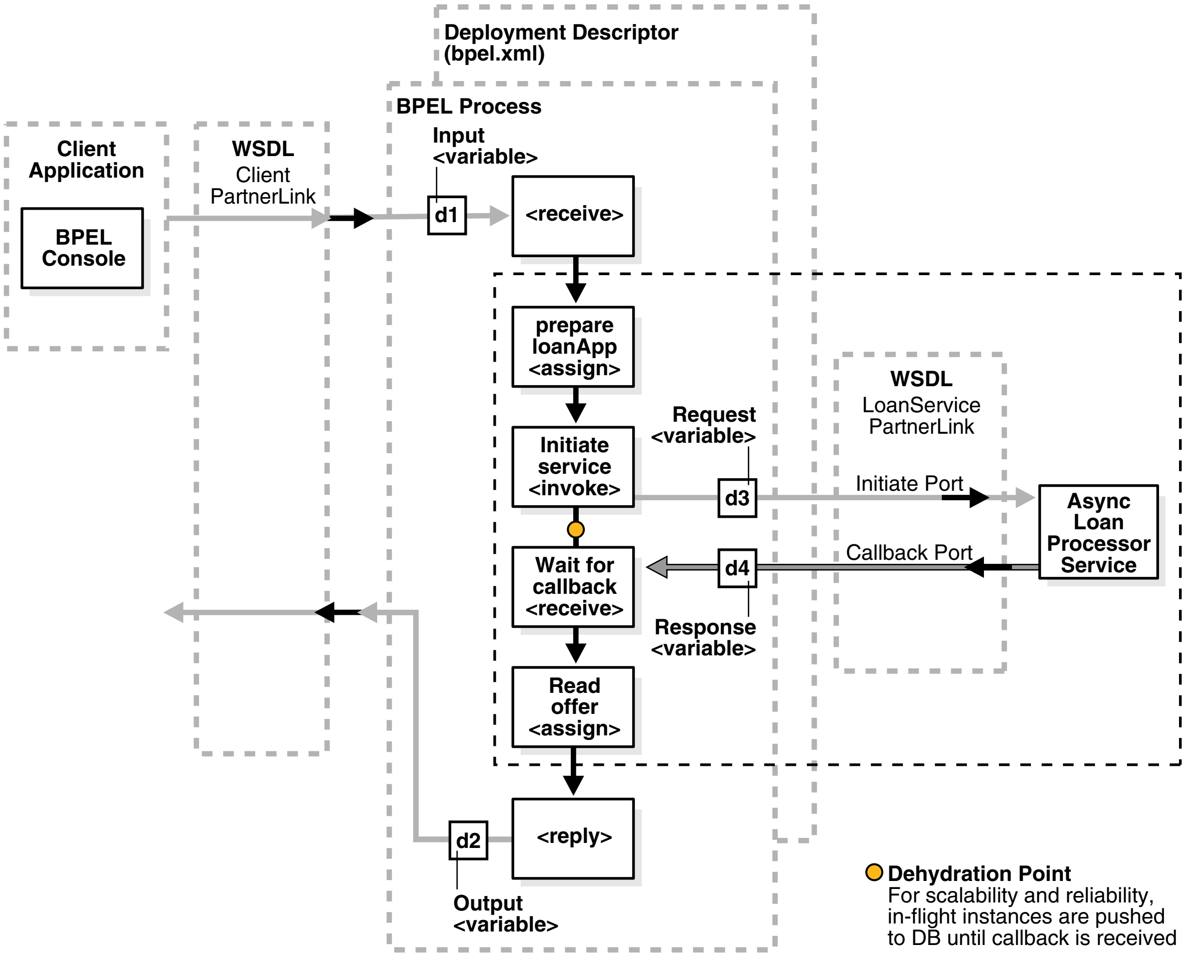
- Introduction
- Use Case
- Understanding Asynchronous Callback Concepts
- Calling an Asynchronous Service
- Questions and Answers
- Summary

Introduction

Figure 6–1 provides an overview of a BPEL process in conversation with an asynchronous loan processor Web service. This Web service processes a client's loan application request and returns a loan offer. Note carefully the boxed area in which the BPEL process uses an invoke activity to initiate the loan application request. The contents of this request are contained in a request variable that is sent to the loan processor Web service. A correlation ID unique to the client and partner link initiating the request is also sent. Since the loan request can take anywhere from a few minutes to a few days to process, it provides an asynchronous interface. The correlation ID ensures that the correct loan offer response is returned to the corresponding loan application requester.

The BPEL process uses a receive activity to wait for a callback response from the Web service. The contents of this response are then stored in a response variable in the BPEL process.

Figure 6–1 Asynchronous Service Invocation



The remaining sections in this chapter provide specific details about the asynchronous functionality shown in Figure 6–1.

See Also: The following sample files for examples of an asynchronous Web service that is not a BPEL process:

- `C:\orabpel\samples\interop\axis\BPELCallingAsyncAXIS`
- *`Oracle_Home`*`\integration\orabpel\samples\interop\axis\BPELCallingAsyncAXIS`

Use Case

United Loan publishes an asynchronous Web service that can take anywhere from several minutes to several days to process a client's loan application request and return a loan offer. This example discusses how to integrate a BPEL process with this asynchronous loan application approver Web service.

This example 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's `.wsdl` file contains the information necessary to request and receive the desired information.

Understanding 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.

See Also: The following files described in this chapter:

If using Eclipse BPEL Designer:

- `C:\orabpel\samples\utils\AsyncLoanService\LoanService.wsdl`
- `C:\orabpel\samples\tutorials\105.AsyncCompositeLoanBroker\ LoanBroker.bpel`
- `C:\orabpel\samples\tutorials\105.AsyncCompositeLoanBroker\bpel.xml`

If using JDeveloper BPEL Designer:

- *`Oracle_Home`*`\integration\orabpel\samples\utils\AsyncLoanService\LoanService.wsdl`
- *`Oracle_Home`*`\integration\orabpel\samples\tutorials\105.AsyncCompositeLoanBroker\LoanBroker.xml`
- *`Oracle_Home`*`\integration\orabpel\samples\tutorials\105.AsyncCompositeLoanBroker\bpel.xml`

partnerLinkTypes for Asynchronous Services

This section examines the Web service's `.wsdl` file, and identifies the sections of the file that enable it to work with BPEL processes.

View the `portType` section of the `LoanService.wsdl` file. This is the loan application approver Web service to which the client submits the loan application request.

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.

```
 <!-- portType implemented by the LoanService BPEL process -->
    <portType name="LoanService">
        <operation name="initiate">
            <input message="tns:LoanServiceRequestMessage"/>
        </operation>
    </portType>

    <!-- portType implemented by the requester of LoanService BPEL process
         for asynchronous callback purposes
         -->
    <portType name="LoanServiceCallback">
        <operation name="onResult">
            <input message="tns:LoanServiceResultMessage"/>
        </operation>
    </portType>
```

View the `partnerLinkType` section of the `LoanService.wsdl` file. The `partnerLinkType` 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.

```
<!-- the LoanService partnerLinkType binds the service and
         requestor portType into an asynchronous conversation.
         -->
    <plnk:partnerLinkType name="LoanService">
        <plnk:role name="LoanServiceProvider">
            <plnk:portType name="tns:LoanService"/>
        </plnk:role>
        <plnk:role name="LoanServiceRequester">
            <plnk:portType name="tns:LoanServiceCallback"/>
        </plnk:role>
    </plnk:partnerLinkType>
```

Note also that two port types are now combined into this single asynchronous BPEL process: `portType="services:LoanService"` of the `invoke` activity and `portType="services:LoanServiceCallback"` of the `receive` activity. Port types are essentially a collection of operations to be performed. For this BPEL process, there are two operations to perform: `initiate` in the `invoke` activity and `onResult` in the `receive` activity.

Calling the Service from BPEL

The following two files are associated with your BPEL process, and define how the process interfaces with the Web service.

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. -->

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

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

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 version="1.0"?>
<bpel-process id="LoanBroker" src="LoanBroker.bpel"
 wsdlLocation="LoanBroker.wsdl">
   <properties id="LoanService">
      <property name="wsdlLocation">
 http://hslattertest-pc:9700/orabpel/default/UnitedLoan/UnitedLoan?wsdl</property>
   </properties>
```

See Also: "Adding a Partner Link for an Asynchronous Service" on page 6-9 for instructions on creating a partner link

Invoke and Receive Activities

View the `variables` and `sequence` sections of the `LoanBroker.bpel` file. Two areas of particular interest are:

- An invoke activity invokes a synchronous Web service (as discussed in Chapter 5, "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 also 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.

```
  <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). 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:

- "Adding an Invoke Activity" on page 6-10
- "Adding a Receive Activity" on page 6-12

Correlations

Because there may 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. There are several supported methods for identifying asynchronous messages to ensure that asynchronous callbacks locate the appropriate client:

- WS-Addressing

  Web Services Addressing (WS-Addressing) is a public specification and is the default correlation method supported by Oracle BPEL Process Manager. No editing of the `.bpel` and `.wsdl` files is required 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 6–2 provides an overview.

Figure 6–2 Callback with WS-Addressing Headers

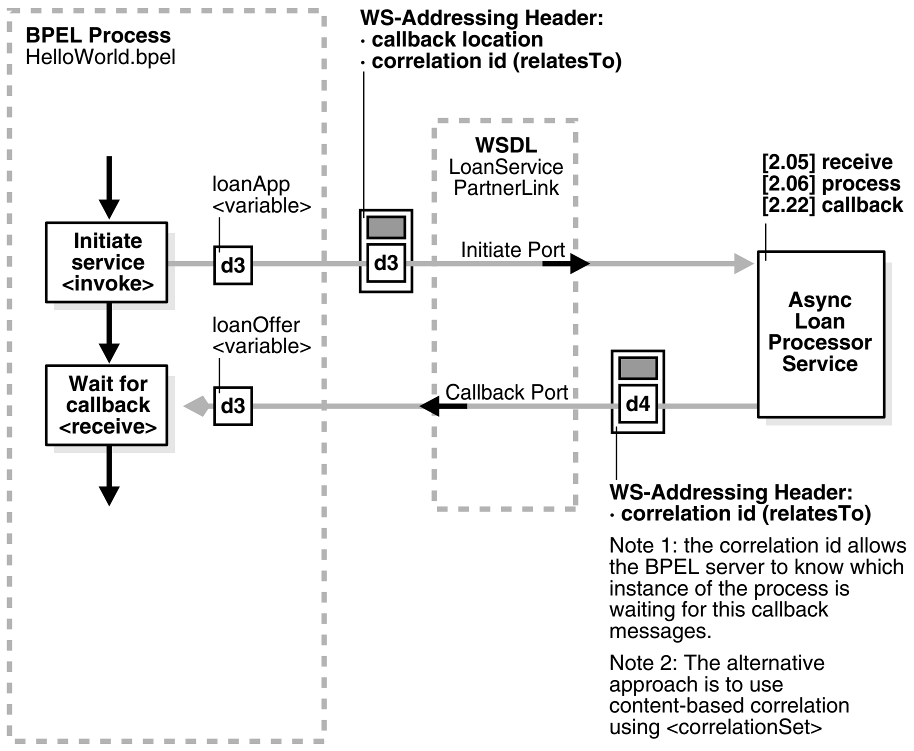

BPEL Process
HelloWorld.bpel
WS-Addressing Header:
· callback location
· correlation id (relatesTo)
WSDL
LoanService
PartnerLink
[2.05] receive
[2.06] process
[2.22] callback
loanApp
<variable>
Initiate service <invoke>
d3
d3
Initiate Port
Async Loan Processor Service
loanOffer
<variable>
Wait for callback <receive>
d3
Callback Port
d4
WS-Addressing Header:
· correlation id (relatesTo)
Note 1: the correlation id allows the BPEL server to know which instance of the process is waiting for this callback messages.
Note 2: The alternative approach is to use content-based correlation using <correlationSet>

Figure 6–2 shows how messages are passed along with WS headers so that the response can be sent to the correct destination.

The example in this chapter uses WS-Addressing for correlation. TCP tunneling can be used for viewing the messages.

- Correlation Sets

  This method uses message content for correlation. You must define correlation sets in your `.bpel` file. Use this method 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`.

WS-Addressing

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.

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

To monitor the SOAP messages, insert a software listener in 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/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.

If you are a JDeveloper BPEL Designer user, you can also use the built-in Packet Monitor to see SOAP messages for both synchronous and asynchronous services.

See Also: The following documentation, which is accessible from `http://www.oracle.com/technology/bpel`:

- *Web Services Addressing (WS-Addressing) Specification* for complete details about WS-Addressing
- *BPEL-TN001: TCP Tunneling the Oracle BPEL Process Manager* for instructions on using TCP tunneling to view SOAP messages

Correlation Sets

Correlation sets are a BPEL mechanism that provides for the correlation of asynchronous messages based on message body contents.

See Also: The following correlation set examples:

- `C:\orabpel\samples\tutorials\109.CorrelationSets` (for Eclipse BPEL Designer)
- `Oracle_Home\integration\orabpel\samples\tutorials\109.CorrelationSets` (for JDeveloper BPEL Designer)

The Reply Activity

The reply activity enables the business process to send a message in reply to a message that was received through a receive. The combination of a receive and a reply forms a request-response operation on the WSDL `portType` for the process.

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

See Also:

- "Reply Activity" on page C-13
- `C:\orabpel\samples\references\Reply` (for Eclipse BPEL Designer)
- *`Oracle_Home`*`\integration\orabpel\samples\references\Reply` (for JDeveloper BPEL Designer)

Dehydration

A dehydration point is also established between the invoke activity and receive activity. (See Figure 6–1 on page 6-2.) Dehydration automatically maintains long-running asynchronous processes (such as this one with the loan application approver Web service) and their current state information in a database while they wait for asynchronous callbacks. Storing the process in a database preserves the process and prevents any loss of state or reliability if a system shuts down or a network problem occurs. This capability increases both BPEL process reliability and scalability and is used to support clustering and failover.

Calling an Asynchronous Service

This section provides an overview of the tasks required to add asynchronous functionality to your BPEL process:

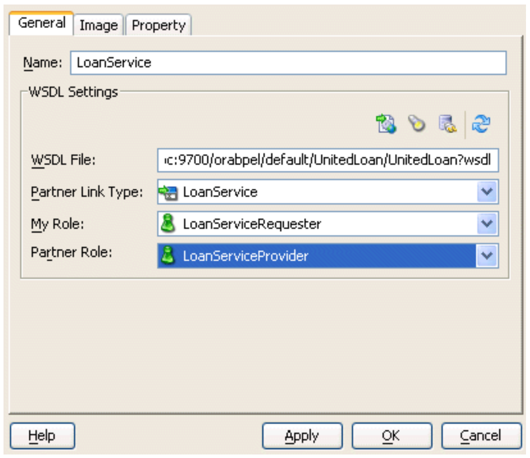
- Adding a Partner Link for an Asynchronous Service
- Adding an Invoke Activity
- Adding a Receive Activity
- Performing Additional Activities

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 **Applications Navigator**.
2. Right-click either side of the BPEL process (in the yellow area).
3. Select **Create Partner Link** from the menu.

   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:


General | Image | Property

Name: LoanService

WSDL Settings

WSDL File: ıc:9700/orabpel/default/UnitedLoan/UnitedLoan?wsdl

Partner Link Type: LoanService

My Role: LoanServiceRequester

Partner Role: LoanServiceProvider

5. Click **OK**.

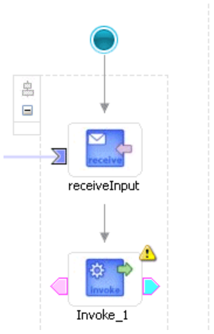
   A new partner link for the loan application approver Web service (United Loan) appears.

   > **See Also:** "partnerLinkTypes for Asynchronous Services" on page 6-4 for conceptual details about partner links

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.

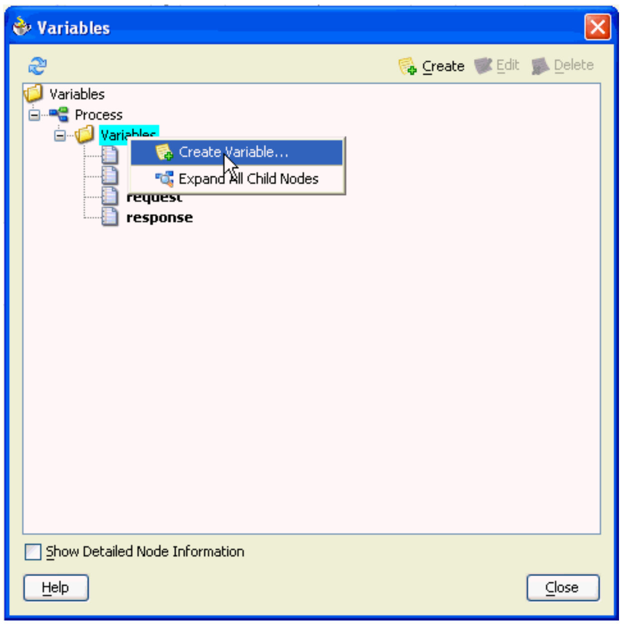
1. Drag an **invoke** activity from the **Component Palette** to after the **receive** activity.



2. Right-click either side of the BPEL process and select **Variables** from the menu.

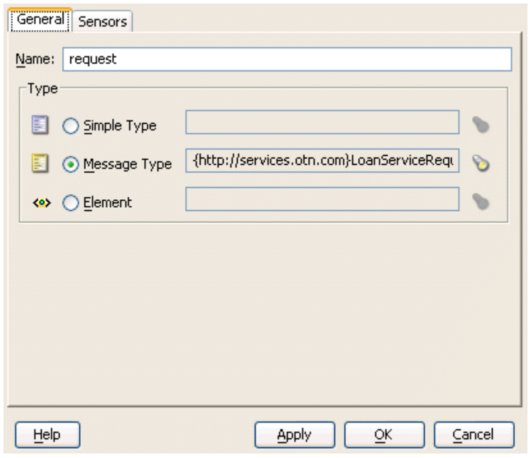
   The Variables window appears.

3. Select **Variables** and right-click, then select **Create Variable**.



The **Create Variable** dialog box appears.

4. Enter the variable name and select the variable type from the options provided:



Name: request

Message Type: {http://services.otn.com}LoanServiceRequ

5. Click **OK**.
6. Return to the Invoke window.
7. Select the **LoanService** partner link from the **Partner Link** list and **initiate** from the **Operation** list of the Invoke window.
8. Select the input variable you created in Step 4.

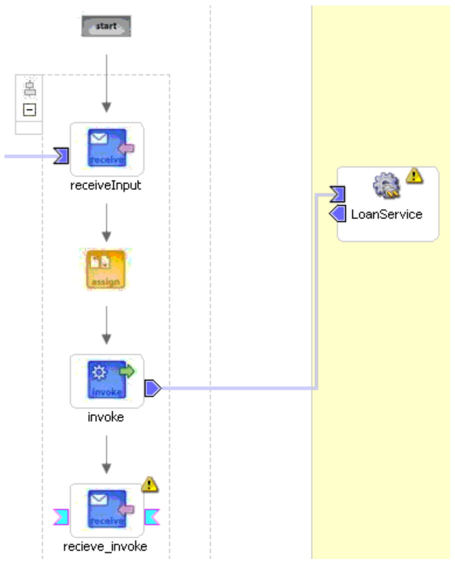
   There is no output variable specified because the output variable is returned in the receive operation. The invoke activity and the global input variable are created.

> **See Also:** "Invoke and Receive Activities" on page 6-5 for conceptual details about the invoke activity

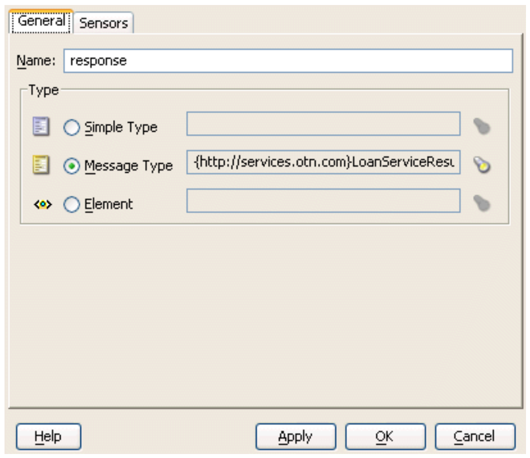
Adding a Receive Activity

Follow these instructions 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. Drag a **receive** activity from the BPEL Palette to after the **invoke** activity you created in "Adding an Invoke Activity" on page 6-10.
2. Change the **receive** activity name to **receive_invoke**.



3. Create a variable by invoking the Create Variable window as you did in Step 2 on page 6-10.
4. Enter the following details:



Name: response

Type

Simple Type

[✓ Message Type] {http://services.otn.com}LoanServiceResu

Element

5. Click **Done**.
6. Return to the Receive window.
7. Select **LoanService** from the **Partner Link** list and **onResult** from the **Operation** list. Do not select the **Create Instance** check box.
8. Select the variable you created in Step 4.

9. Click **OK**.

   The **receive** activity and the output variable are created. Note that 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.

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.

Questions and Answers

What does the createInstance attribute do?

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.

```
<!-- receive input from requestor -->
<receive name="receiveInput" partnerLink="client"
         portType="tns:LoanBroker"
         operation="initiate" variable="input"
         createInstance="yes"/>
```

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.

7 Parallel Flow

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.

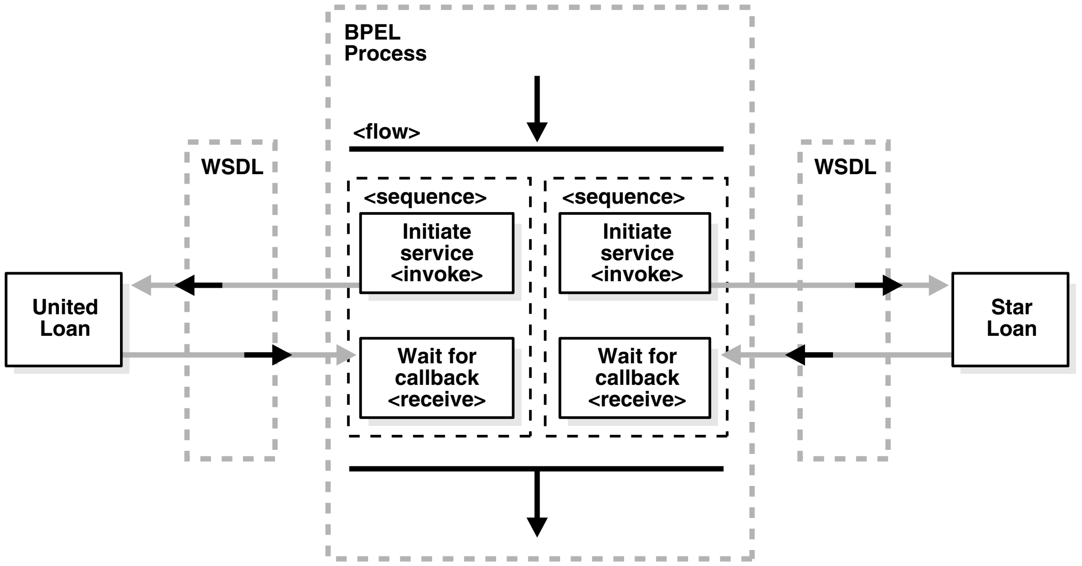
This chapter contains the following topics:

- Introduction
- Use Case
- Understanding Parallel Flow Concepts
- Defining a Parallel Flow
- The flowN Activity
- Summary

Introduction

Figure 7–1 provides an overview of the BPEL process performing a parallel flow to retrieve loan offers from two different Web services. 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.

Figure 7–1 Parallel Flow Invocation


BPEL Process
<flow>
WSDL
WSDL
<sequence>
<sequence>
Initiate service <invoke>
Initiate service <invoke>
United Loan
Star Loan
Wait for callback <receive>
Wait for callback <receive>

Use Case

Since the United Loan service can take up to several days to return a loan offer and you also want to collect a loan offer from StarLoan, define your BPEL process so both tasks run in parallel.

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

See Also: The following samples:

- `C:\orabpel\samples\tutorials\106.ParallelFlows` (for Eclipse BPEL Designer)
- *`Oracle_Home`*`\integration\orabpel\samples\tutorials\106.ParallelFlows` (for JDeveloper BPEL Designer)

Understanding Parallel Flow Concepts

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

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).

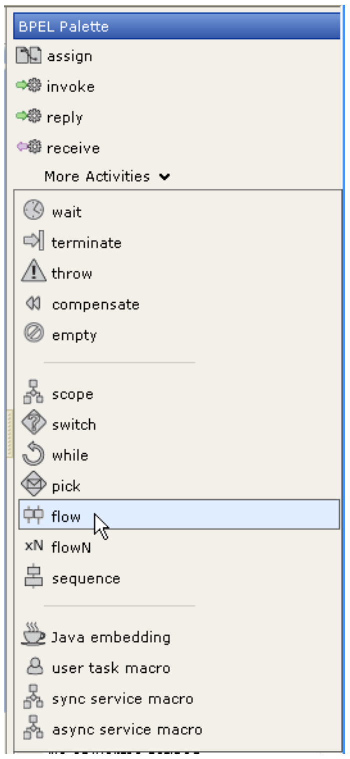
```
<flow name="flow-1">
   <sequence>
          <scope name="UnitedLoan">
             <sequence>
                  <invoke name="invoke-2" partnerLink="unitedLoan"
                   portType="services:LoanService" operation="initiate"
                   inputVariable="loanApplication"/>
         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

The above example shows two sequences, but the flow can have many sequences.

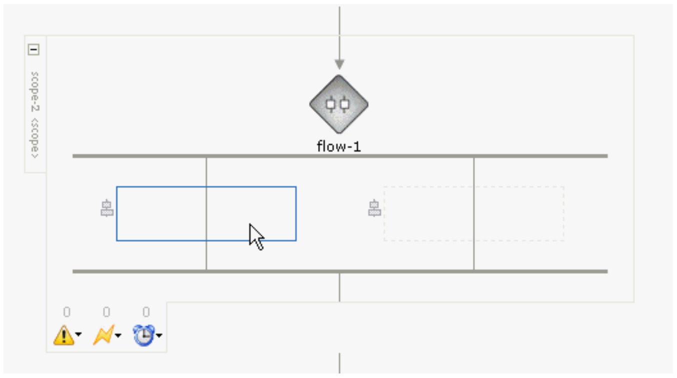
Follow these instructions to create a flow activity and a global input variable named `request`. This activity initiates an asynchronous BPEL process activity with a loan offer Web service (United Loan). The loan offer service uses the request input variable to receive the loan request from the client.

This example shows how to create a flow activity in Eclipse BPEL Designer.

1. Click **BPEL Designer** > **Process Map**.
2. Drag a **flow** activity from the **BPEL Palette** into a **scope** activity.



3. The **flow** activity includes two branches, each with a box for functional elements. Populate these boxes as you do a **scope** activity, either by building a function with **BPEL Palette** activities or by dragging activities from your **Process Map** into the boxes.



See Also: The following documentation for examples of creating flow activities in JDeveloper BPEL Designer:

- *Oracle BPEL Process Manager Order Booking Tutorial*
- *Oracle BPEL Process Manager Quick Start Guide*
- "Flow Activity" on page C-6

The flowN Activity

In the `flow` activity the number of parallel branches are determined by the BPEL code. 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 runtime 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 `FlowN` activity performs activities on an arbitrary number of data elements. As the number of elements changes, the BPEL process adjusts accordingly.

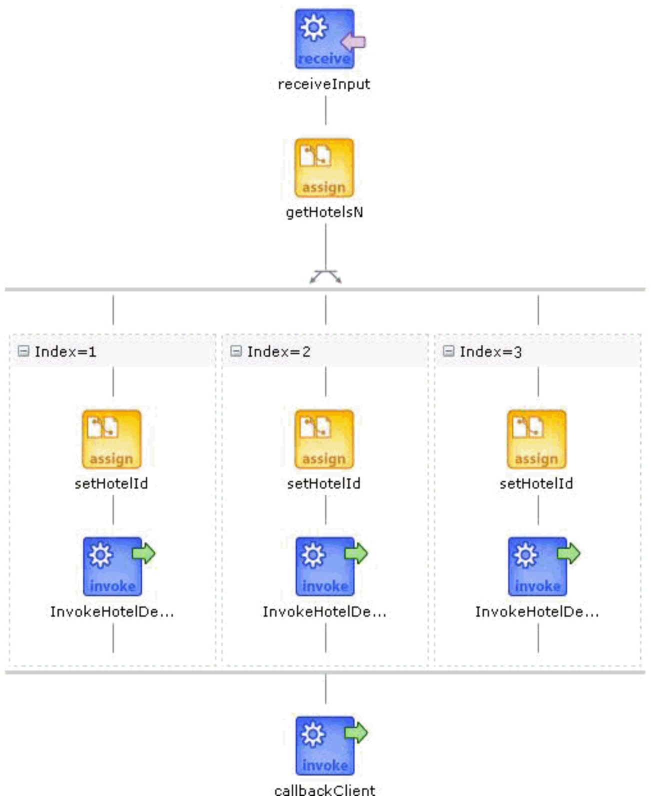
The branches created by flowN perform the same activities, but use different data. Each branch will use the index variable to look up input variables. The index variable can be used in the XPATH expression to acquire the data specific for that branch.

For example, if there is an array of data the BPEL process uses a `count` activity 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 a 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. So if the array contain 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.

`FlowN` can use data from other sources as well, including data obtained from web services.

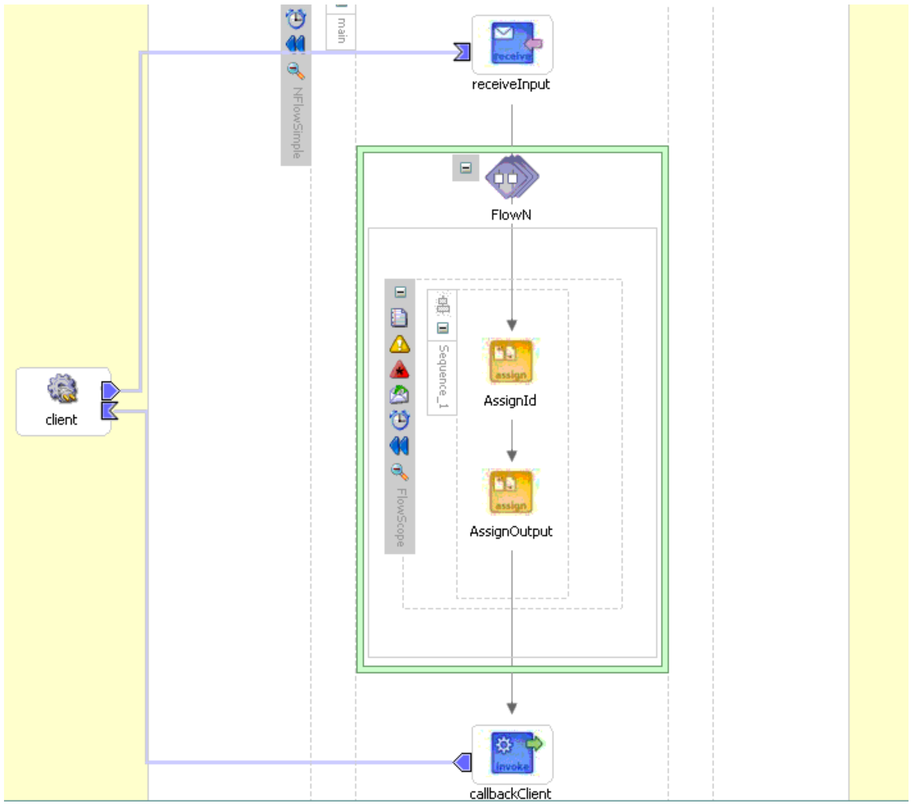
The following figure shows a console view of a flowN activity that looks up three hotels. This is different from the view because instead of showing the BPEL process, it shows how the process has actually executed. In this case there are three hotels, but the number of branches changes to match the number of hotels available:

Figure 7–2 A Console View of the Execution of a FlowN activity

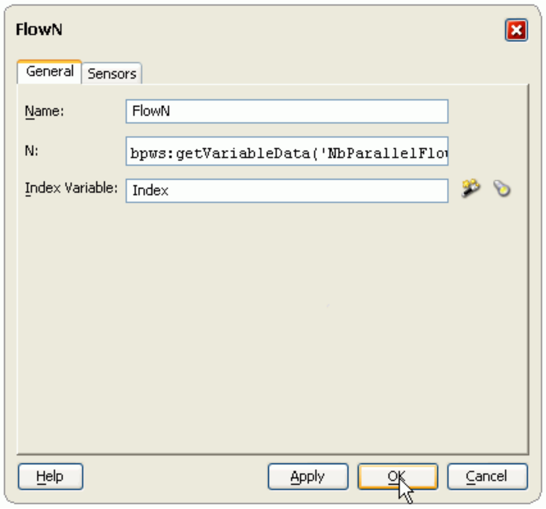


In the JDev version of the BPEL Designer, a `flowN` activity appears as follows:

Figure 7–3 FlowN Activity Setup in the Diagram View



By double clicking on the `flowN` activity, the flowN Wizard appears:



Name:	FlowN
N:	bpws:getVariableData('NbParallelFlo
Index Variable:	Index

This wizard allows you to set the name of the flowN activity, enter an expression for calculating the value of N, and defining the index variable.

BPEL Code Example of the FlowN Activity

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

```
  <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"
    -->
```

The `receive` activity calls the client partner link to get the information flowN needs to define N and look up hotel information.

```
    <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
       noded passed in.
       For lisibility, an intermediate varaible 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:NflowHot
elsProcessRequest/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 here. 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.
            -->
```

The following copy rule uses the index variable to concatenates the hotel entries into a list.

```
        <assign name="setHotelId">
          <copy>
            <from
expression="bpws:getVariableData('inputVariable','payload',concat('/client:NflowHo
telsProcessRequest/client:ListOfHotels/client:HotelName[',bpws:getVariableData('In
dex'),']'))"/>
```

```
            <to variable="InvokeHotelDetailInputVariable" part="payload"
query="/ns2:hotelInfoRequest/ns2:id"/>
          </copy>
        </assign>
```

Using the hotel information, an invoke 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 procees doesn't do anything with the retrieved inforamtion.
        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 a executed
        in parallel, not sequentially.
        -->
      </sequence>
    </bpelx:flowN>
```

Finally, the BPEL process sends detailed information on each hotel to the client partner link:

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

Summary

This chapter shows how to create a parallel flow using the `flow` activity to perform multiple tasks simultaneously. This BPEL process performs two asynchronous callbacks in parallel, which can take considerably less time than performing the two callbacks in series. Another activity called `flowN` allows the BPEL PM to use data to spawn the necessary number of parallel flows at run time, 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.

8 Conditional Branching

This chapter describes conditional branching. Conditional branching introduces decision points to control the flow of execution of a BPEL process.

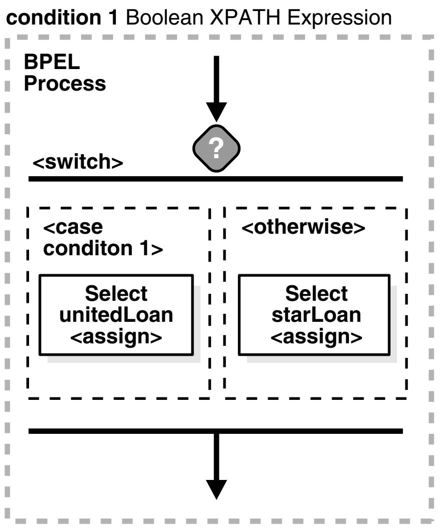
This chapter contains the following topics:

- Introduction
- Use Case
- Understanding Conditional Branching Concepts
- Conditional Branching
- The While Activity
- Summary

Introduction

Figure 8–1 provides an overview of the BPEL process performing a conditional branching activity to select a path (or branch) based on two pieces of data; in this case, two loan offers.

Figure 8–1 Conditional Branching



Use Case

The BPEL process has 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.

Understanding Conditional Branching Concepts

BPEL applies logic to make choices through conditional branching. A number of branches are set up, and each branch has a condition 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 the first true branch is executed.

Conditional Branching

In Chapter 7, "Parallel Flow", the flow activity of the BPEL process gathered two loan offers at the same time, but did not do any comparison 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.

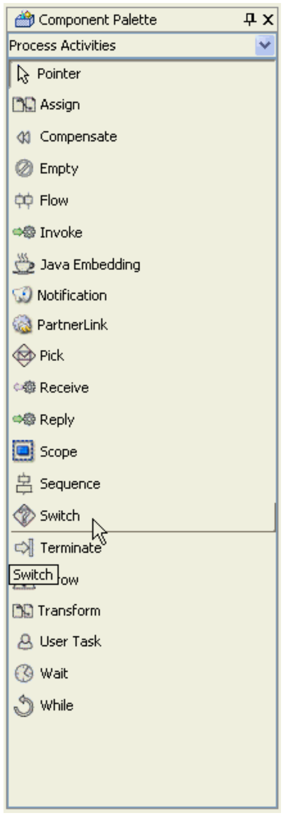
A switch activity, like a flow activity, has multiple branches. In this example, there are only two branches. The first branch is executed if a case condition containing an XPath Boolean expression is met; otherwise, the second branch is executed.

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

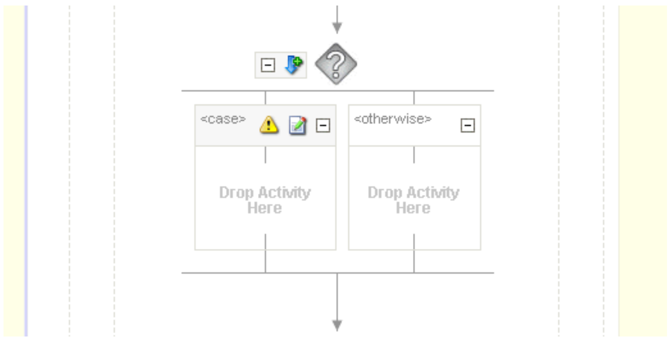
Adding a Switch Activity

To add a switch activity to your BPEL flow in JDeveloper BPEL Designer:

1. Drag a **switch** activity from the **Component Palette** into your BPEL flow.



2. The **switch** activity includes two branches by default, each with a box for functional elements.



3. Click the first branch to highlight it, right-click, and select **Edit** from the menu.

   The Switch Case window appears.

4. Enter an XPath Boolean expression in the **Expression** field. If this expression is true, then the first branch is executed. If it is false, then the second branch is executed. You may also include comments in the **Name** field.

In the case of the MyLoanFlow tutorial, the two loan offers are stored in the global variables `loanOffer2` and `loanOffer1`. Each loan offer contains several pieces of data; one of those pieces of data is the APR of the loan offer. The BPEL flow chooses the loan with the lowest APR. Therefore, if `loanOffer1` has a higher APR, then the first branch selects `loanOffer2` by assigning `loanOffer2`'s payload to `selectedLoanOffer's` payload. However, if `loanOffer1` does *not* have a lower APR than `loanOffer2`, then the `otherwise` case assigns `loanOffer1`'s payload to `selectedLoanOffer's` payload.

The XPath Boolean expression for this case is:

```
bpws:getVariableDate('loanOffer1','payload','/loanOffer/APR') >
bpws:getVariableData('loanOffer2','payload','/loanOffer/APR')
```

See Also: The following documentation for examples of creating switch activities in JDeveloper BPEL Designer:

- *Oracle BPEL Process Manager Order Booking Tutorial*
- *Oracle BPEL Process Manager Quick Start Guide*
- "Switch Activity" on page C-15
- `Oracle_Home\integration\orabpel\samples\references\Switch` (for JDeveloper BPEL Designer)
- `c:\orabpel\samples\references\Switch` (for Eclipse BPEL Designer)

The While Activity

The while looping conditional activity enables you to repeat an activity until a certain success criteria has been 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 a 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).

The following syntax is from the *Business Process Execution Language for Web Services Specification*:

```
<while condition="bool-expr" standard-attributes>
   standard-elements
   activity
</while>
```

See Also: The following documentation for examples of defining a while activity in JDeveloper BPEL Designer:

- *Oracle BPEL Process Manager Order Booking Tutorial*
- "While Activity" on page C-20
- `Oracle_Home\integration\orabpel\samples\references\While` (for JDeveloper BPEL Designer)
- `c:\orabpel\samples\references\While` (for Eclipse BPEL Designer)

Summary

This chapter discusses the concepts and procedures for creating a 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.

9 Fault Handling

Fault handling allows a BPEL process to deal with 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:

- Introduction
- Use Case
- Defining a Fault Handler
- Taxonomy of BPEL Faults
- Using the Scope Activity
- Throwing Internal Faults
- Returning External Faults
- Fault Handler Within a Scope
- Compensation
- The Terminate Activity
- Catching Run-Time Faults Example
- Eclipse BPEL Designer Example
- Summary

Introduction

Web services occasionally return errors or faults instead of the data normally expected. There are two kinds of faults: business faults and run time faults. Business faults are the result of a problem with the information, for example when a social security number is not found in the database. Run time faults are the result of problems within the BPEL process or the Web service themselves, as when data cannot be copied properly because the variable name is incorrect.

Use Case

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

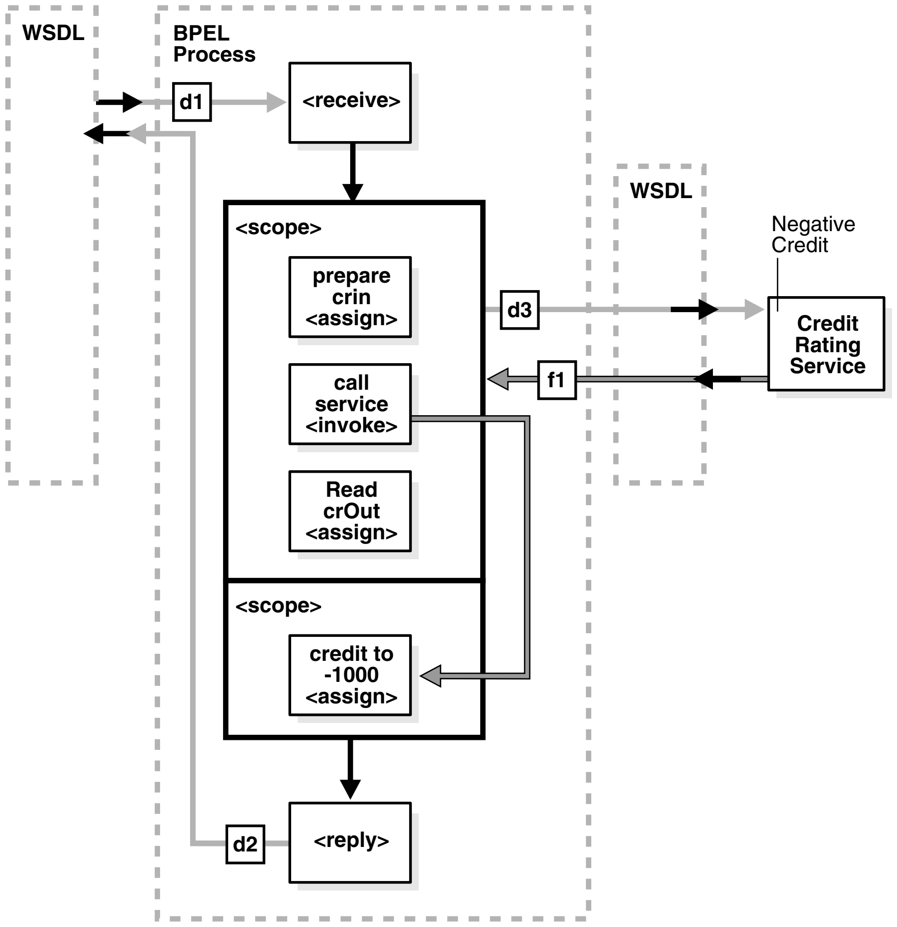
See Also: The following samples:

- `C:\orabpel\samples\tutorials\107.Exceptions` (for Eclipse BPEL Designer)
- *`Oracle_Home`*`\integration\orabpel\samples\tutorials\107.Exceptions` (for JDeveloper BPEL Designer)

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. In this example, the credit rating variable is set at -1000, as shown in Figure 9–1.

Figure 9–1 Fault Handling



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

```
<faultHandlers>
     <catch faultName="services:NegativeCredit" faultVariable="crError">
      <assign name="assign-2">
         <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 selecting Web services for the BPEL process, determine the possible faults that may be returned and set up a fault handler for each one.

Taxonomy of BPEL Faults

A BPEL fault has a fault name called a `Qname` and a possible `messageType`. There are two categories of faults in BPEL: business fault and run-time fault.

Business faults are application specific faults that are generated when there is a problem with the information being processed. 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.

> **See Also:** *BPEL Technical Note #007, Managing BPEL Run-time Exceptions*, at: `http://www.oracle.com/technology/products/ias/bpel/htdocs/orabpel_technotes.tn007.html` for more detailed information on BPEL fault codes

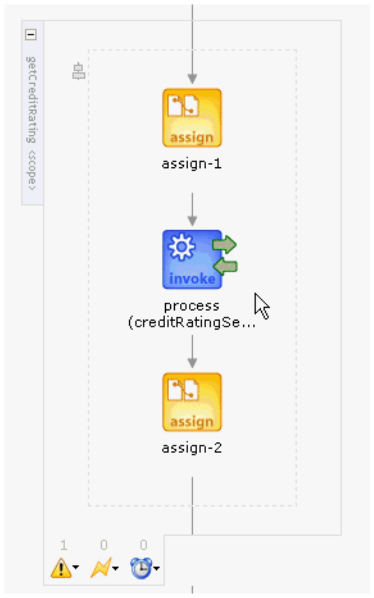
Using the Scope Activity

The scope activity is one of the key BPEL development tools. A scope is 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 simplifies your BPEL flow by grouping functional structures together, allowing you to collapse them into what appears to be a single element in the BPEL designer (either JDeveloper BPEL Designer or Eclipse BPEL Designer).

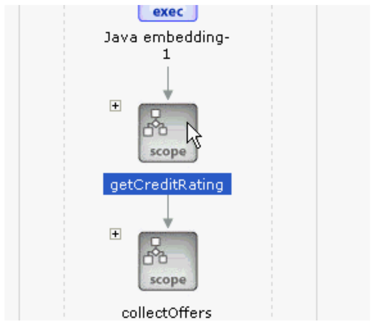
The following code example shows a scope activity. In this case, the process for getting a credit rating based on a customer's social security number has been placed inside a scope named `getCreditRating`. At the level of the BPEL code, this identifies functional blocks of code and sets them apart visually. In the BPEL designer (either Eclipse BPEL Designer or JDeveloper BPEL Designer), the activities contained inside the scope can be collapsed into a single visual element, or expanded when necessary.

```
<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>
```

In the user interface, the **getCreditRating** scope appears as the following when expanded:

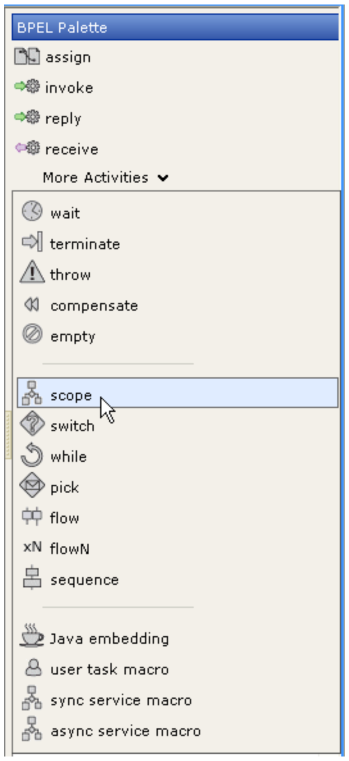


The collapsed scope appears as a single design element, as follows:

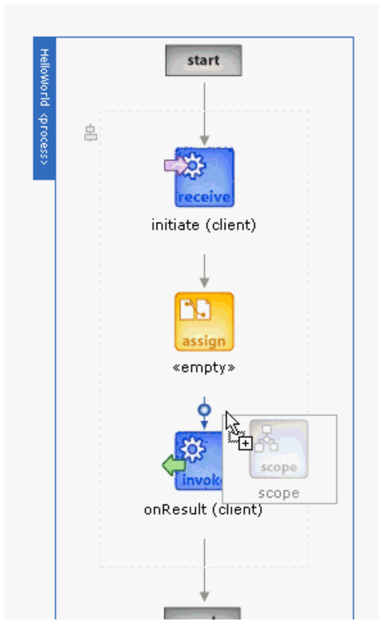


To add a scope (for this example, using Eclipse BPEL Designer):

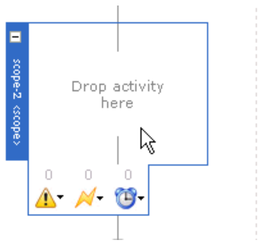
1. From the **BPEL Palette** (by default, in the upper right corner of the **Designer** in the **Process Map** view), click **More Activities** to open the full palette menu.



2. Click and drag a **scope** into the **Process Map**.



3. Open the **scope** by double-clicking it or by single-clicking the **+** sign.



4. Drag activities from the **BPEL Palette** to build the function within the scope. You can also drag the activities composing a function from your existing flow in the **Process Map** and drop them into the scope.

> **See Also:** The following documentation for examples of creating scope activities in JDeveloper BPEL Designer:
>
> - *Oracle BPEL Process Manager Order Booking Tutorial*
> - *Oracle BPEL Process Manager Quick Start Guide*
> - "Scope Activity" on page C-15

Throwing Internal Faults

A BPEL application can generate fault messages as well as receive them. The throw activity has three elements: its name, the name of the `faultName`, and the `faultVariable`. 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`. Here is a code sample of a throw activity:

```
<throw name="delay" faultName="fault-1" faultVariable="fVar"/>
     <invoke name="invokeStockQuoteService" partnerLink="StockQuoteService"/>
     <assign>
        <copy>
           <from variable="response" part="result" query="/result"/>
           <to variable="output" part="payload" query="/tns:result"/>
        </copy>
     </assign>
```

You can see 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.

> **See Also:** The following documentation for examples of creating throw activities:
>
> - "Throw Activity" on page C-17
> - `*Oracle_Home*\integration\orabpel\samples\references\Throw` (for JDeveloper BPEL Designer)
> - `c:\orabpel\samples\references\Throw` (for Eclipse BPEL Designer)

Returning External Faults

A BPEL process can also 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:

```
<reply partnerlinke="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. A useful way of employing this activity is to make it 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 8, "Conditional Branching" for more information on setting up the conditional structure

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.

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. A scope 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 specify specific fault names in the catch activity to respond in a specific way to an individual fault. There is also a catchAll activity that catches any faults not already handled by name-specific catch activities.

> **See Also:** The following documentation for examples of creating fault handling:
>
> - *Oracle BPEL Process Manager Order Booking Tutorial*
> - *Oracle BPEL Process Manager Quick Start Guide*
> - "Catch Activity" on page C-4
> - `Oracle_Home\integration\orabpel\samples\references\Catch` (for JDeveloper BPEL Designer)
> - `c:\orabpel\samples\references\Catch` (for Eclipse BPEL Designer)

The Empty Activity at runtime

There is often a need to use an activity that does nothing (for example, when a fault needs to be caught and suppressed). The empty activity enables you to insert a no-op instruction into a business process. The syntax to use is minimal:

```
 <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 had been performed.

Consider the following example:

```
<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 matches the type of bar and whose name is not `x:foo` is processed by the second `catch`. All other faults are processed by the default `catchAll` handler.

See Also:

- "Catch Activity" on page C-4
- "Empty Activity" on page C-6

Compensation

Compensation is 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.

The compensation handler can be invoked 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 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.

Note that in case an invoke activity has a compensation handler defined inline, the name of the activity is the name of the scope to be used in the compensate activity.

```
<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 BPEL4WS. This activity can be used 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:

```
<compensate scope="RecordPayment"/>
```

If a scope being compensated by name was nested in a loop, the instances of the compensation handlers in the successive iterations are invoked 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, causes this default behavior to be invoked explicitly. This is useful when an enclosing fault or compensation handler must perform additional work, such as updating variables or sending external notifications, in addition to performing default compensation for inner scopes. Note that the compensate activity in a fault or compensation handler attached to scope `S` causes the default-order invocation of compensation handlers for completed scopes directly nested within `S`. The use of this activity can be mixed with any other user-specified behavior except the explicit invocation of `<compensate scope="Sx"/>` for scope `Sx` nested directly within `S`. Explicit invocation of compensation for such a scope nested within `S` disables the availability of default-order compensation, as expected.

See Also: "Compensate Activity" on page C-5

The Terminate Activity

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. Note that this does not give you any notification of the status of the BPEL process. If you are going to use the terminate activity, first program notifications to the interested parties.

```
<terminate standard-attributes>
    standard-elements
  </terminate>
```

See Also: The following documentation for examples of creating terminate activities:

- *Oracle BPEL Process Manager Order Booking Tutorial*
- *Oracle BPEL Process Manager Quick Start Guide*
- "Terminate Activity" on page C-16
- `c:\orabpel\samples\references\Terminate` (for Eclipse BPEL Designer)
- `Oracle_Home\integration\orabpel\samples\Terminate` (for JDeveloper BPEL Designer)

Catching Run-Time Faults Example

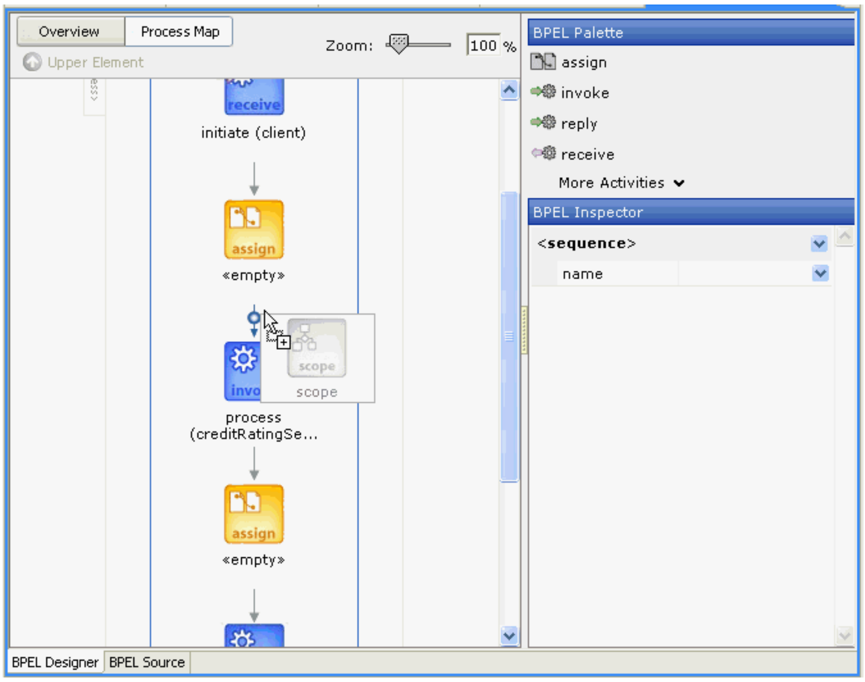
The following procedure shows you how to use the provided examples to generate a fault and define a fault handler to catch it. In this case, you modify a WSDL file to generate a fault, and create a catch attribute to catch it.

1. Import `RuntimeFault.wsdl` into your process WSDL (under the `Oracle_Home\integration\orabpel\system\xmllib` directory).
2. Declare a variable with `messageType “bpelx:RuntimeFaultMessage”`.
3. Catch it using `<catch faultName="bpelx:remoteFault"  | “bpelx:bindingFault” faultName="varName">`.

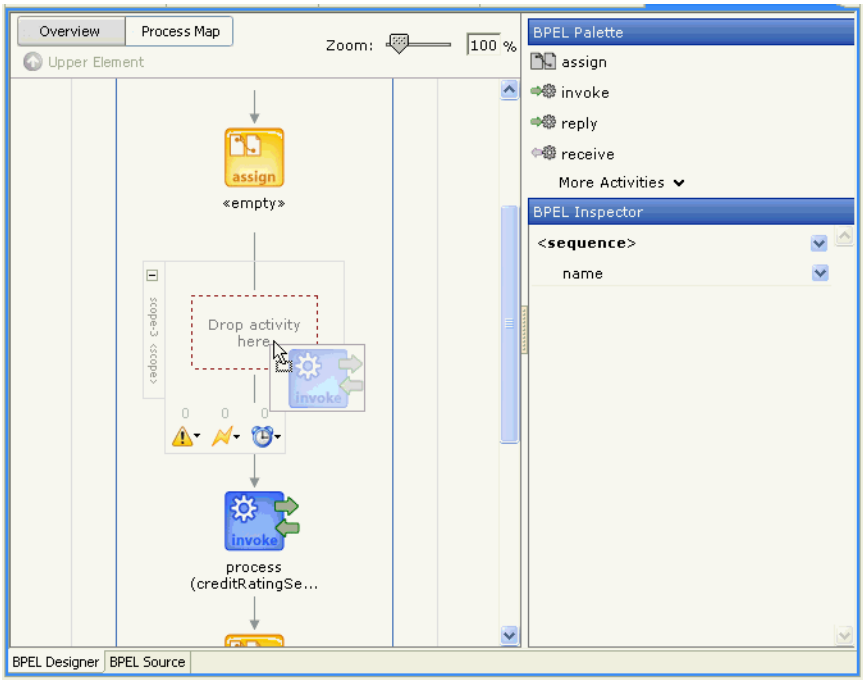
Eclipse BPEL Designer Example

To define a fault handler in Eclipse BPEL Designer, follow these steps:

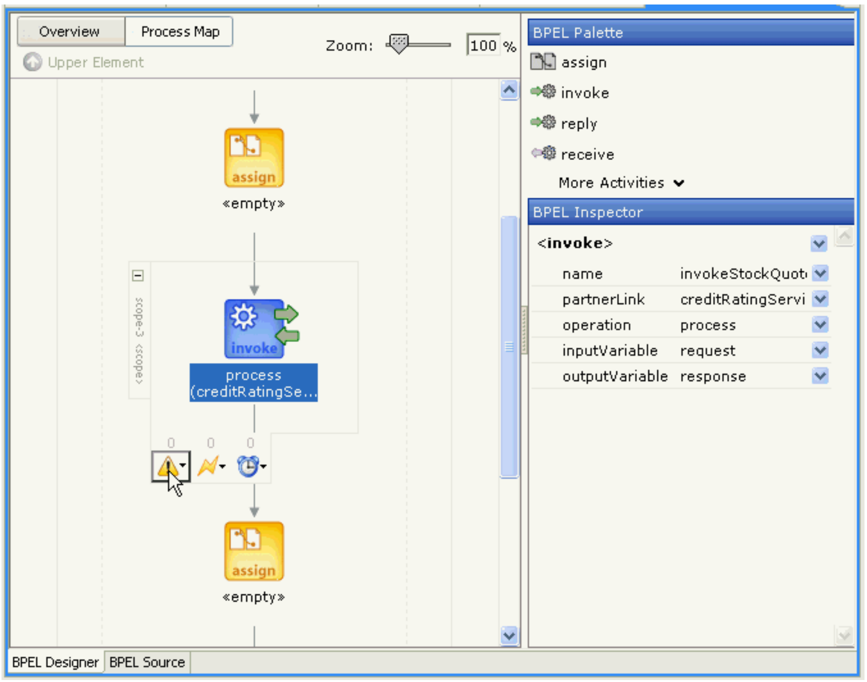
1. Wrap the activity in a scope by selecting a scope from the **BPEL Palette**, and dragging it into your BPEL process just above the **invoke** credit rating activity.



2. Expand the **scope** and drag the **invoke** credit rating activity into the scope.

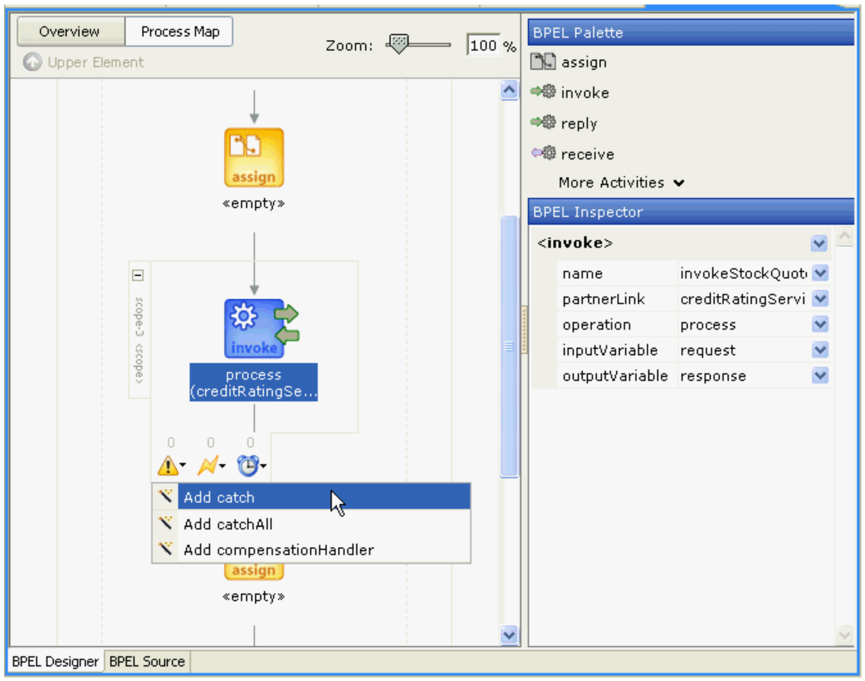


At the bottom of the **scope** area, there are three icons: a caution sign, a lightning bolt, and a clock.



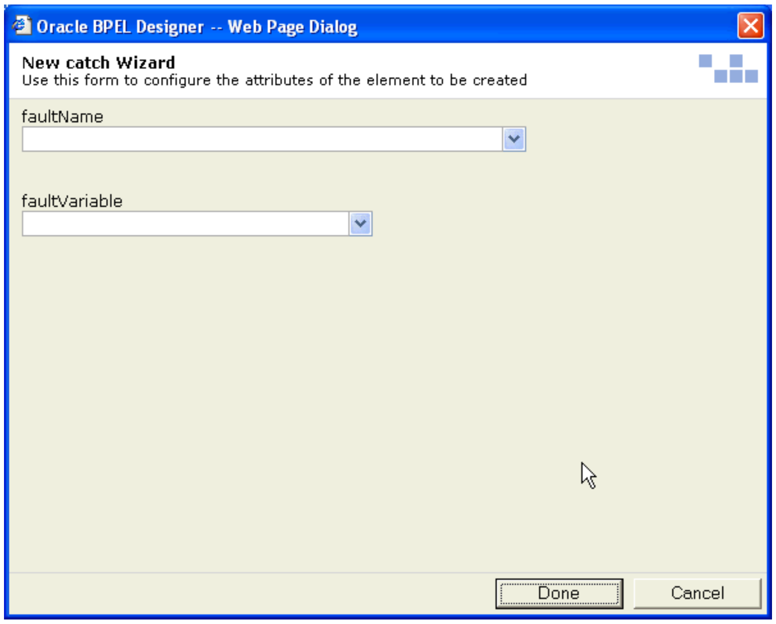
<invoke>	
name	invokeStockQuot
partnerLink	creditRatingServi
operation	process
inputVariable	request
outputVariable	response

3. Click the caution sign, and select **Add catch**.



name	invokeStockQuot
partnerLink	creditRatingServi
operation	process
inputVariable	request
outputVariable	response

The New Catch Wizard window appears:

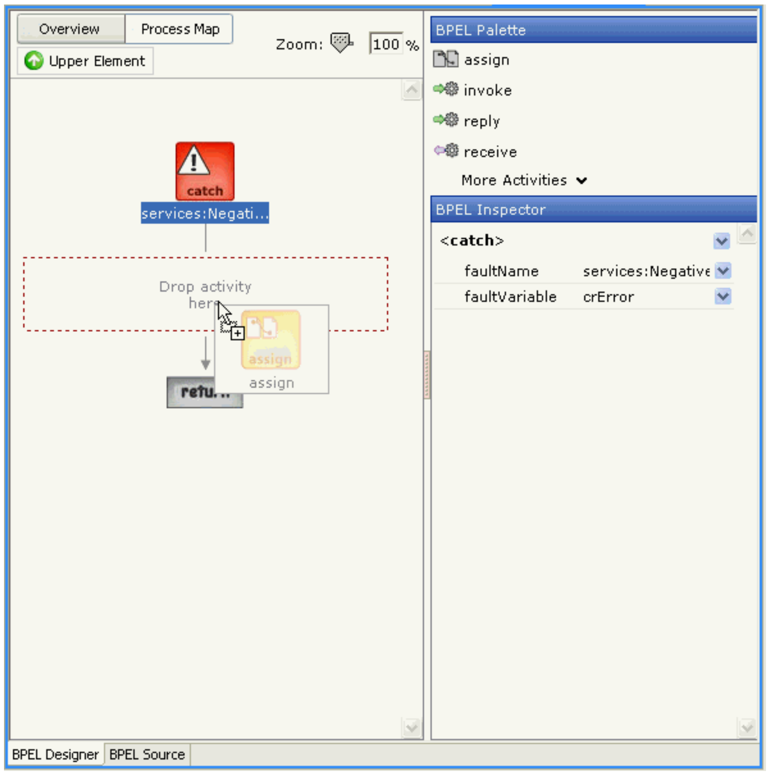


New catch Wizard
Use this form to configure the attributes of the element to be created

4. Select the fault name and the fault variable from the drop-down menus provided.

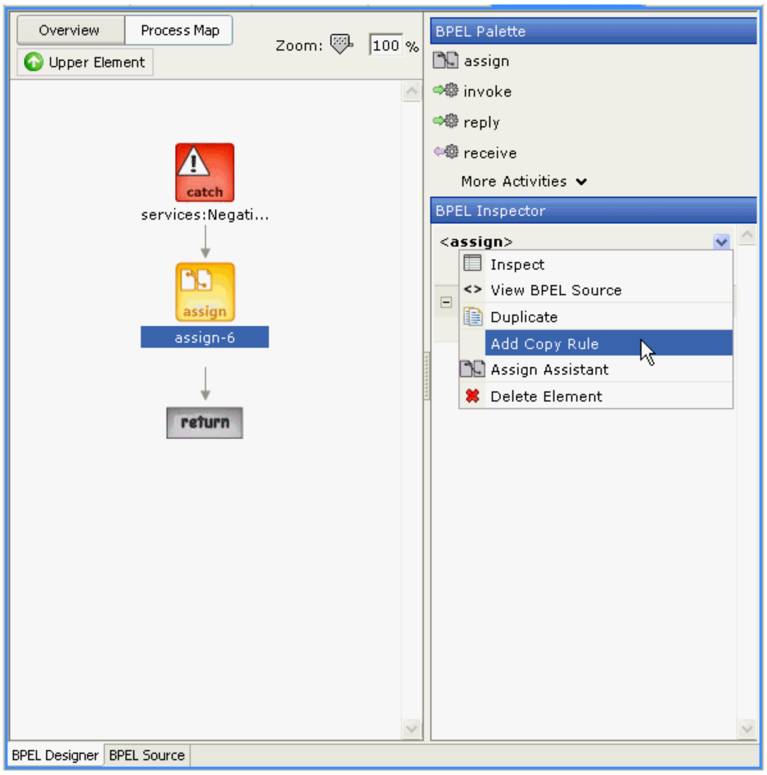
   This defines where the fault comes from and what fault activates this fault handler.

5. Drag an **assign** activity into the BPEL process below the **catch** activity.



6. Add copy rules to the **assign** activity.

   This defines what the BPEL flow does when it receives the defined fault.



7. Click the **Upper Element** button to return to the **Process Map** view.

See Also: The following documentation for examples of defining fault handling:

- *Oracle BPEL Process Manager Order Booking Tutorial*
- *Oracle BPEL Process Manager Quick Start Guide*
- "Catch Activity" on page C-4
- `Oracle_Home\integration\orabpel\samples\references\Catch` (for JDeveloper BPEL Designer)
- `c:\orabpel\samples\references\Catch` (for Eclipse BPEL Designer)

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.

10 Incorporating Java/J2EE Code in BPEL Processes

This chapter demonstrates how you can embed sections of Java code into a BPEL process.

This chapter contains the following topics:

- Introduction
- Use Case
- Using Java Code with WSIF Binding
- Using Java Code Wrapped as a SOAP Service
- Embedding Java Code in BPEL
- JDeveloper BPEL Designer Example
- Summary

Introduction

This chapter demonstrates 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.

Use Case

You have a customer entity bean that retrieves a social security number based on an e-mail id. How can you invoke that bean from within the BPEL process?

Using Java Code with WSIF Binding

Java code can be used from the BPEL process if the Java application has a BPEL compatible interface. Two compatible interfaces are as follows:

- WSIF binding
- Wrapping as a SOAP service

WSIF Binding is the most common way of using Java code in a BPEL process.

WSIF binding enables a BPEL process to invoke an EJB through native J2EE protocol (local or 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, but has less interoperability (each application server needs a specific binding) and currently less tool support than SOAP services. The binding must currently be written by hand.

See Also:

- `C:\orabpel\samples\tutorials\702.Bindings` for Eclipse BPEL Designer or `*Oracle_Home*\integration\orabpel\samples\tutorials\702.Bindings` for examples of WSIF bindings for EJB, HTTP, and Java. Bindings must be written to match the application server.
- `C:\orabpel\samples\demos\BankTransferDemo\BankTransferFlow` (for Eclipse BPEL Designer)
- `*Oracle_Home*\integration\orabpel\samples\demos\BankTransferDemo\BankTransferFlow` (for JDeveloper BPEL Designer)

Using Java Code Wrapped as a SOAP Service

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:

- Sacrifices performance, because interactions are constantly being mapped back and forth between the Java code and the SOAP wrapper

- 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)

Embedding Java Code in BPEL

Another way to make use of 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, only fairly small segments of code can be incorporated.

The bpelx:exec Tag

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 JTA transaction context.

Java exceptions are converted into BPEL faults and put into a BPEL process.

The Java snippet can propagate its JTA transaction to session and entity beans that it calls.

XML Facade

An XML facade can be used to simplify DOM manipulation. Oracle BPEL Process Manager provides a lightweight JAXB-like Java object model on top of XML (called a facade). XML facade provides a Java bean-like front end for an XML document/element that has a schema. Facade classes can provide easy manipulation of the XML document and element in Java programs.

The facade classes are generated using the `schemac` tool shipped with Oracle BPEL Process Manager. You can run `schemac *.wsdl / *.xsd` to generate the facades from WSDL or XSD files.

Oracle ships a sample that showcases the use of facade classes in Java binding. This is similar to your use case. This sample is in the following directories:

- `c:\orabpel\samples\tutorials\702.Bindings\JavaBinding` (for Eclipse BPEL Designer)
- *`Oracle_Home`*`\integration\orabpel\samples\tutorials\702.Bindings\JavaBinding` (for JDeveloper BPEL Designer)

Review the sample to see how it is done. Some points of interest in that sample are as follows:

1. In `build.xml`, you can see a task called `schemac`. This is basically the same as doing `schemac` at the command line. This generates the source of the facade classes.
2. The Java binding provider class `HelperService.java` is under `702.Bindings\JavaBinding\src\com\otn\services`. It has one method:

   ```
   public CommentsType addComment(CommentsType payload, CommentType comment)
   ```

   which uses the facade classes `CommentsType` and `CommentType`.
3. In `HelperService.wsdl`, a Java binding service is defined. See the `format:typeMapping` section of Java binding:

```
    <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>
```

which maps XML types to the corresponding facade classes.

See Also: "schemac" on page 19-24

bpelx:exec Built-in Methods

A set of built-in methods enable you to read and update scope variables, instance metadata, and audit trails. Table 10–1 describes these methods.

Table 10–1 Built in Methods for <bpelx:exec>

Method Name	Description
`Object lookup( String name )`	JNDI access
`Locator getLocator( )`	BPEL PM Locator
`long getInstanceId( )`	Unique id associated with each instance
`String setTitle( String title ) / String getTitle()`	Title of this instance
`String setStatus( String status ) / String getStatus()`	Status of this instance
`void setIndex( int i, String value ) / String getIndex( int i )`	Six indexes can be used for search
`void setPriority( int priority ) / int getPriority()`	Priority
`void setCreator( String creator ) / String getCreator()`	Who initiated this instance
`void setCustomKey( String customKey ) / String getCustomKey()`	Second primary key
`void setMetadata( String metadata ) / String getMetadata ()`	Metadata for generating lists
`String getPreference( String key )`	Access preference defined in `bpel.xml`
`void addAuditTrailEntry(String message, Object detail)`	Add an entry to the audit trail
`void addAuditTrailEntry(Throwable t)`	Access file stored in the suitcase
`Object getVariableData(String name) throws BPELFault`	Access and update variables stored in the scope
`Object getVariableData(String name, String partOrQuery) throws BPELFault`	
`Object getVariableData(String name, String part, String query)`	
`void setVariableData(String name, Object value)`	
`void setVariableData(String name, String part, Object value)`	

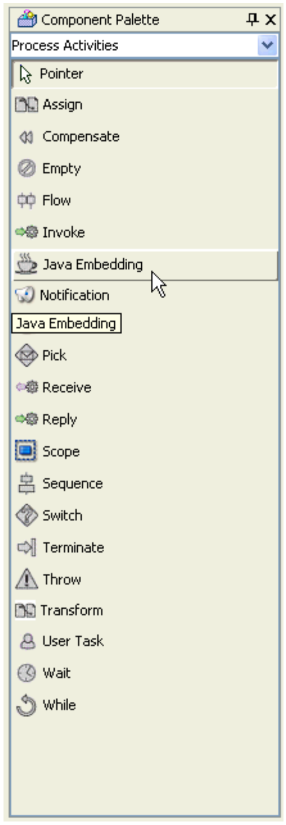
Table 10–1 (Cont.) Built in Methods for <bpelx:exec>

Method Name	Description
`void setVariableData(String name, String part, String query, Object value)`	

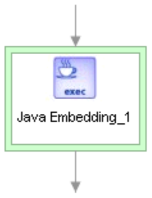
JDeveloper BPEL Designer Example

JDeveloper BPEL Designer enables you to add the `bpelx:exec` activity, and copy the code snippet into a dialog box.

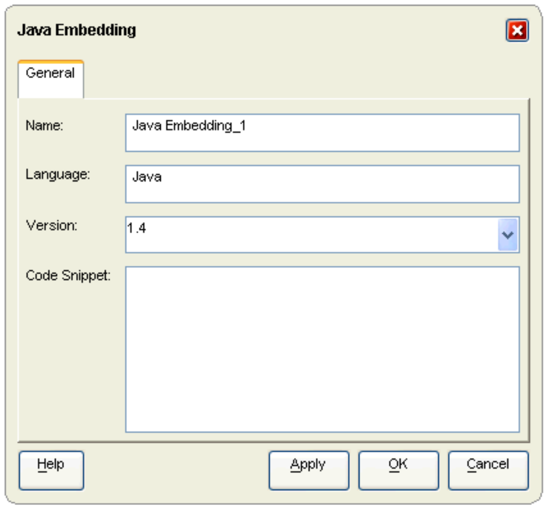
1. Select the **Java Embedding** activity (with the coffee cup icon) from the **Component Palette**.



2. Drag the activity into your BPEL process.



3. Double-click the **Java Embedding** activity.
4. The Java Embedding window appears. Enter (or cut and paste) the Java code into the **Code Snippet** field:



Name: Java Embedding_1

Language: Java

Version: 1.4

Code Snippet:

See Also: "Java Embedding Activity" on page C-9 for additional details about this activity

Summary

This chapter demonstrates how you can embed sections of Java code into a BPEL process. One way to integrate an existing Java component into a BPEL process is 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.

A set of built-in methods enable the `bpelx:exec` snippet to read and update variables, change instance meta data, and throw faults.

An XML facade can be used within code to simplify DOM manipulation.

11 Events and Timeouts

This chapter describes how to use events and timeouts. Since asynchronous Web services can take a long time to return a response, your 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.

This chapter contains the following topics:

- Introduction
- Use Case
- The pick Activity
- The Wait Activity
- JDeveloper BPEL Designer Example
- Synchronous Processes
- Summary

Introduction

Since asynchronous Web services can take a very long time to return a response, your 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. The pick activity allows the BPEL flow to deal with an either/or case; that is, continue when either condition A (receive a response) or condition B (wait 12 hours) is satisfied. The pick activity can also be used for other either/or choices as well.

Use Case

In this example, the BPEL process is programmed to wait 1 minute for a Star Loan response. If Star Loan does not respond in one minute, then the United Loan offer is automatically selected. 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.

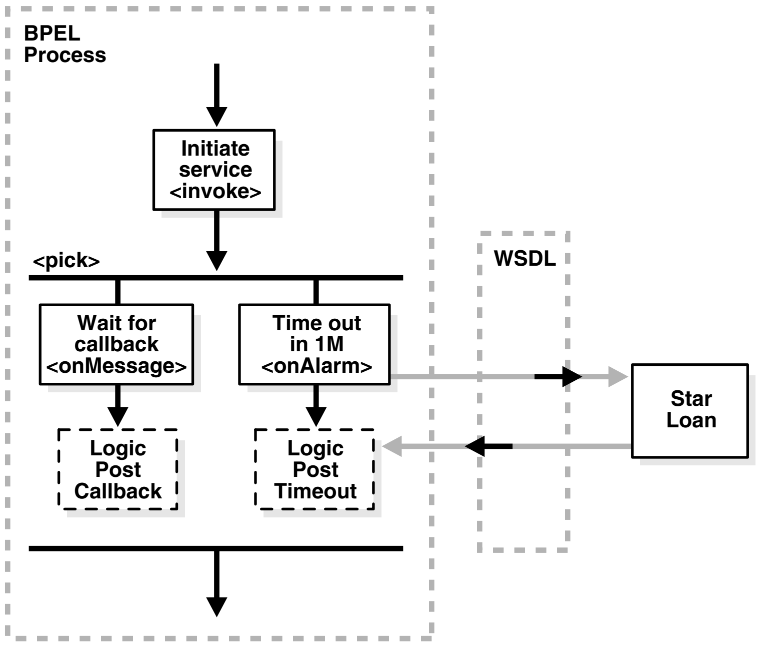
See Also: The following sample files:

- `C:\orabpel\samples\tutorials\108.Timeouts` (for Eclipse BPEL Designer)
- *`Oracle_Home`*`\integration\orabpel\samples\tutorials\108.Timeouts` (for JDeveloper BPEL Designer)

The pick Activity

The pick activity provides two branches, each one with a condition. The branch that has its condition satisfied first is executed. For the loan offer timeout example, one branch's condition is to receive the loan offer, and the other branch's condition is to wait a certain amount of time. Figure 11–1 provides an overview.

Figure 11–1 The pick Activity



The following code segment defines the `pick` activity for this operation:

```
  <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 timesout -->
       <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>
```

The `pick` activity contains two branches, the `onMessage` branch and the `onAlarm` branch. The `onMessage` branch contains the code for receiving a reply from the Star Loan service, while the `onAlarm` branch contains the code for a timeout, in this case after one minute. Whichever branch completes first is executed; the other branch is not.

The `onMessage` code is the same as the code for receiving a response from the Star Loan service before the timeout was added.

The `onAlarm` code has a time associated with it. This time is defined as `PT1M` in this case, which means to wait one minute. In this item, `S` stands for seconds, `M` for minute, `H` for hour, `D` for day, and `Y` for year. Therefore, a time limit of `1` year, `3` days, and `15` seconds is entered 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`

See Also: The following samples:

- `C:\orabpel\samples\references\Pick` (for Eclipse BPEL Designer)
- `*Oracle_ Home*\integration\orabpel\samples\references\Pick` (for JDeveloper BPEL Designer)
- "Pick Activity" on page C-11

The Wait Activity

The wait 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.

```
<wait (for="duration-expr" | until="deadline-expr") standard-attributes>
    standard-elements
  </wait>
```

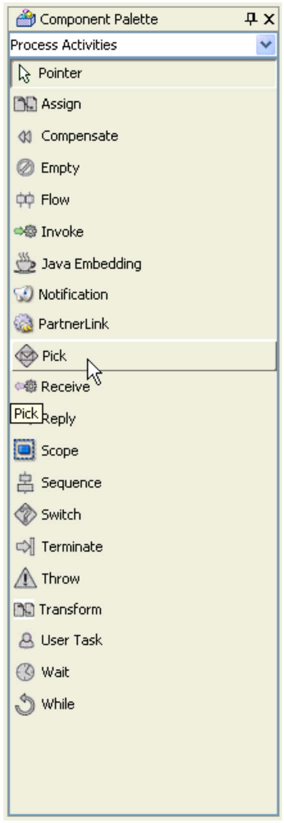
See Also: The following documentation for examples of defining a wait activity in JDeveloper BPEL Designer:

- *Oracle BPEL Process Manager Order Booking Tutorial*
- "Wait Activity" on page C-19

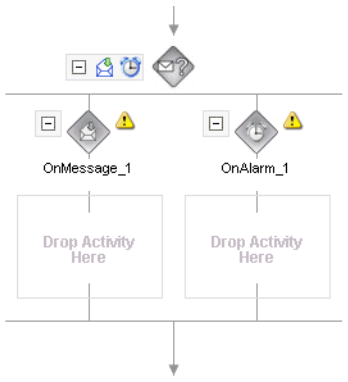
JDeveloper BPEL Designer Example

To define a timeout in JDeveloper BPEL Designer, follow these steps:

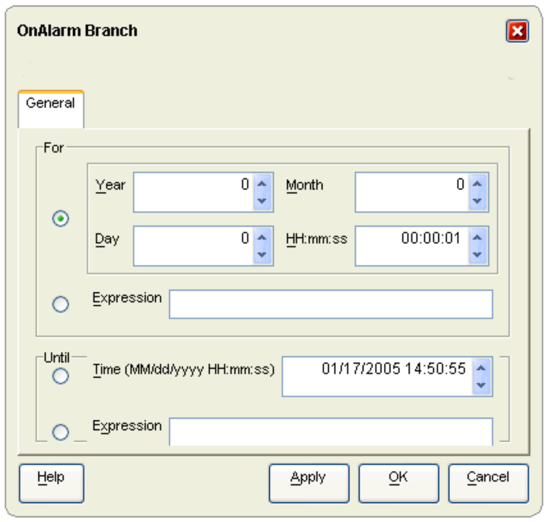
1. Select the **pick** attribute from the **Component Palette**, and drag it into the appropriate part of your BPEL process. In this case, place it just before the **receive** activity for the Star Loan service.



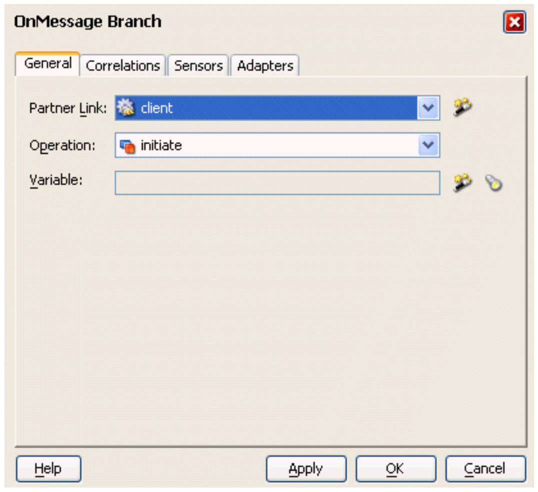
The **pick** activity includes the **onMessage** and **onAlarm** branches.



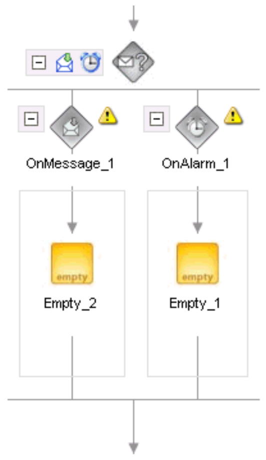
2. Edit the **onAlarm** activity so that the time limit is **1** minute instead of **1** hour by double-clicking the **onAlarm** branch.



3. Press **OK** when complete.
4. Double-click the **onMessage** activity, and edit its attributes to receive the response from the loan service.



5. Put empty **assign** activities in the **Drop activity here** areas below the **onMessage** and **onAlarm** branches. In other instances, this space can contain additional logic and processing.



Synchronous Processes

For synchronous processes that connect to a remote database, you must increase the `syncMaxWaitTime` timeout property in the *`Oracle_Home`*`\integration\orabpel\domains\default\config\domain.xml` file:

```
<property id="syncMaxWaitTime">
        <name>Delivery result receiver maximum wait time</name>
        <value>60000000</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.
        <p/>
        The default value is 60 seconds.]]>
        </comment>
    </property>
```

Note: For Eclipse BPEL Designer, `domain.xml` is located in the `c:\orabpel\domains\default\config` directory.

Summary

Instead of performing multiple operations at the same time as with the flow attribute, the pick activity enables you 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 the Star Loan service, and the other branch is a timeout set at one minute.

12 Invoking a BPEL Process

This chapter shows how a JAVA/JSP application can call a BPEL process in order to perform functions or use services.

This chapter contains the following topics:

- Introduction
- Use Case
- Sending Messages to a BPEL Process from a Java/JSP Application
- Summary

Introduction

In order for BPEL to be useful, it must be able to interact with Web interfaces. This chapter shows how a JAVA/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, BPEL processes deployed on Oracle BPEL Process Manager are also made available to clients through a Java API. This chapter describes how to invoke BPEL processes, both synchronous and asynchronous, through either SOAP or Java.

See Also: The tutorial at `http://www.oracle.com/technology/products/ias/bpel/pdf/orabpel-Tutorial7-InvokingBPELProcesses.pdf`

Use Case

In this example, you can log onto a Web site, enter a social security number, and get 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 files:

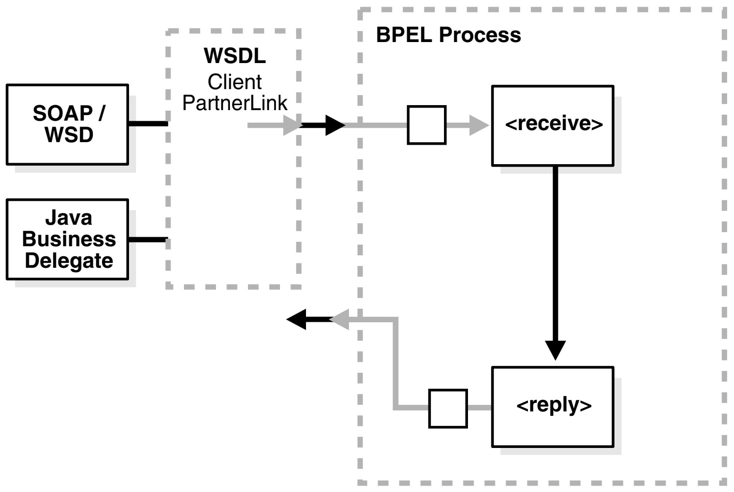
- `C:\orabpel\samples\tutorials\102.InvokingProcesses` (for Eclipse BPEL Designer)
- *`Oracle_Home`*`\integration\orabpel\samples\tutorials\102.InvokingProcesses` (for JDeveloper BPEL Designer)

Sending Messages to a BPEL Process from a Java/JSP Application

A BPEL process can be invoked as a Web Service through a WSDL/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.

Figure 12–1 illustrates how an application interacts with a BPEL process through a client partner link, using one of a number of possible protocols.

Figure 12–1 Application Interaction with a BPEL Process



Invoking a BPEL Process with the Generic Java API

BPEL processes can be invoked using a Java API provided through a stateless session bean interface. The API is slightly different depending on whether you are invoking a two-way operation (which has both input and output messages) or a one-way operation (which just has an input message and returns no result). As such, two code examples are provided in the samples directory, one for invoking the CreditRating BPEL process, which provides a synchronous service with a two-way process operation, and one for initiating the Hello World BPEL process, which is an asynchronous service with a one-way BPEL initiate operation. These two samples use a few common building blocks, and are discussed in more detail later.

Connecting to Oracle BPEL Process Manager with the Locator Class

To support a flexible client interface without being affected by server clustering and other production configuration details, a `com.oracle.bpel.client.Locator` class is provided for connecting to Oracle BPEL Process Manager, authenticating if required, and obtaining handles to services provided by that 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 an `com.oracle.bpel.client.dispatch.IDeliveryService` instance. This instance can invoke or initiate BPEL processes deployed on Oracle BPEL Server:

```
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

Locator locator = new Locator("default", "bpel", null);

IDeliveryService deliveryService = (IDeliveryService)locator.lookupService
     (IDeliveryService.SERVICE_NAME );
```

Note: The domain password can be changed from Oracle BPEL Admin Console. If the password has been changed from the default password, then the instance must be updated with the correct password.

See Also: "Changing Oracle BPEL Admin Console Password" on page 19-8

Passing XML Messages through Java

Because all Web services, including BPEL processes, accept and return XML messages, any Java API using Web services needs a way to pass XML data through Java. Oracle BPEL Process Manager has a client class, `com.oracle.bpel.client.NormalizedMessage`, which allows you 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 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 );
```

In practice, you activate `NormalizedMessages` more dynamically. For full documentation of the `NormalizedMessage` class, see the Oracle BPEL Process Manager Javadocs in:

- `C:\orabpel\docs\apidocs` (for Eclipse BPEL Designer)
- *`Oracle_Home`*`\integration\orabpel\docs\apidocs` (for JDeveloper BPEL Designer)

Invoking a Two-Way Operation through the Java API

Once a delivery service has been instantiated, it can initiate the BPEL process with a `NormalizedMessage` XML message. To invoke a two-way Web service operation that returns a result synchronously, use one of the `IDeliveryService.request()` methods. This method is overloaded and you see the Javadoc for all the available versions of it. However, here the `request()` method has the following signature:

```
public NormalizedMessage request(java.lang.String processId,
                                 java.lang.String operationName,
                                 NormalizedMessage message)
                          throws java.rmi.RemoteException
```

A code example of using this API to invoke the `CreditRatingService` BPEL process is provided with the Oracle BPEL Process Manager samples and is shown below.

Full JSP source:

```
<%@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>
<head>
<title>Invoke CreditRatingService</title>
</head>
<body>
<%
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!<br>" );
out.println( "Credit Rating is " + payload.get("payload") );
%>
```

See Also: The following samples:

- `C:\orabpel\samples\tutorials\102.InvokingProcesses\jsp\invokeCreditRatingService.jsp`
- *`Oracle_Home`*`\integration\orabpel\samples\tutorials\102.InvokingProcesses\jsp\invokeCreditRatingService.jsp`

Invoking a One-Way Operation through Java API

The procedure for invoking a one-way BPEL operation through the Java API is very similar, except that you use the `IDeliveryService.post()` method (which is also overloaded). These 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). In the following code example, the `post` method is exactly the same as the `request()` shown above, except that it returns void:

From the Javadoc for `com.oracle.bpel.client.dispatch.IDeliveryService`:

```
public void post(java.lang.String processId,
               java.lang.String operationName,
               NormalizedMessage message)
         throws java.rmi.RemoteException
```

Selected JSP source:

```
<%@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);
out.println( "BPELProcess HelloWorld initiated!" );
%>
```

See Also: The following samples:

- `C:\orabpel\samples\tutorials\102.InvokingProcesses\jsp\invokeHelloWorld.jsp`
- *`Oracle_Home`*`\integration\orabpel\samples\tutorials\102.InvokingProcesses\jsp\invokeHelloWorld.jsp`

Retrieving Status/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. In some cases, this is not an issue. For example, the LoanFlowPlus BPEL demonstration application (located in `C:\orabpel\samples\demos\LoanDemoPlus` for Eclipse BPEL Designer or *`Oracle_Home`*`\integration\orabpel\samples\demos\LoanDemoPlus` for JDeveloper BPEL Designer) avoids this issue by informing users of process progress through a user task where you can manually approve the final loan offer. In some cases, the process sends some sort of notification, such as an e-mail message or a JMS message, when it completes. Also, a Java client can poll for the result of an asynchronous BPEL process. In this case, the client needs a handle to fetch status information for a particular instance. While the `post()` method does not automatically return such a handle, it does support the client specifying a conversation ID that can be any unique identifier that the client can later use to identify a specific instance and retrieve status information for it. See the Javadocs for the `com.oracle.bpel.client.NormalizedMessage` class to see the specific field name for the conversation ID and other properties, which can be 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.

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.

This is a more advanced use case. Contact Oracle Support Services for more information on how to accomplish this in your specific environment.

> **See Also:** The following samples for examples of how to send e-mail or JMS messages from BPEL processes:
>
> If using Eclipse BPEL Designer:
>
> - `C:\orabpel\samples\tutorials\116.SendEmails`
> - `C:\orabpel\samples\tutorials\118.JMSService\buyer`
>
> If using Eclipse BPEL Designer:
>
> - *Oracle_Home*`\integration\orabpel\samples\tutorials\116.SendEmails`
> - *Oracle_Home*`\integration\orabpel\samples\tutorials\118.JMSService\buyer`

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. These APIs are remotable, however, and can be used through RMI from a remote application server. The RMI client code is different based 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/environment.

Invoking a BPEL Process with the Web Service/SOAP Interface

Once deployed to Oracle BPEL Server, a BPEL process 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 available. 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/SOAP interface and a business delegate Java interface. The Java business delegate interface allows Java/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.

13 Interaction Patterns

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:

- Introduction
- 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
- Third-Party Interactions
- Summary

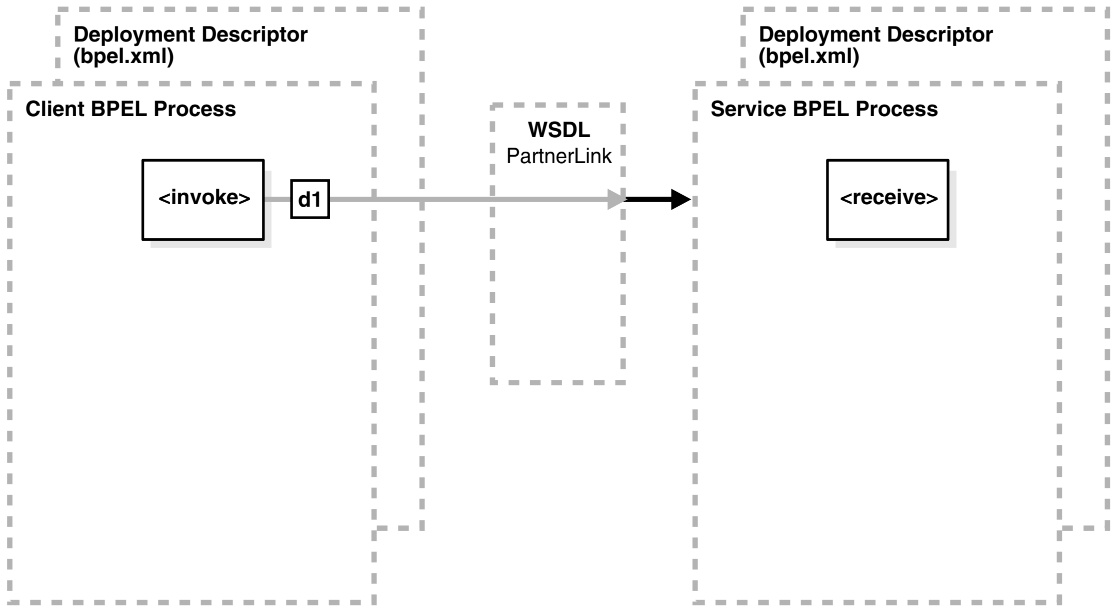
Introduction

This chapter identifies common interaction patterns between a BPEL process and another application, and shows the best use practices for each.

One-Way Message

A one-way message, or fire and forget, is where the client sends a message to the service, and the service does not need to reply. Figure 13–1 provides an overview.

Figure 13–1 One-Way Message



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.

Note: All BPEL interactions require a valid partner link. For the rest of the examples, assume that there is a valid partner link.

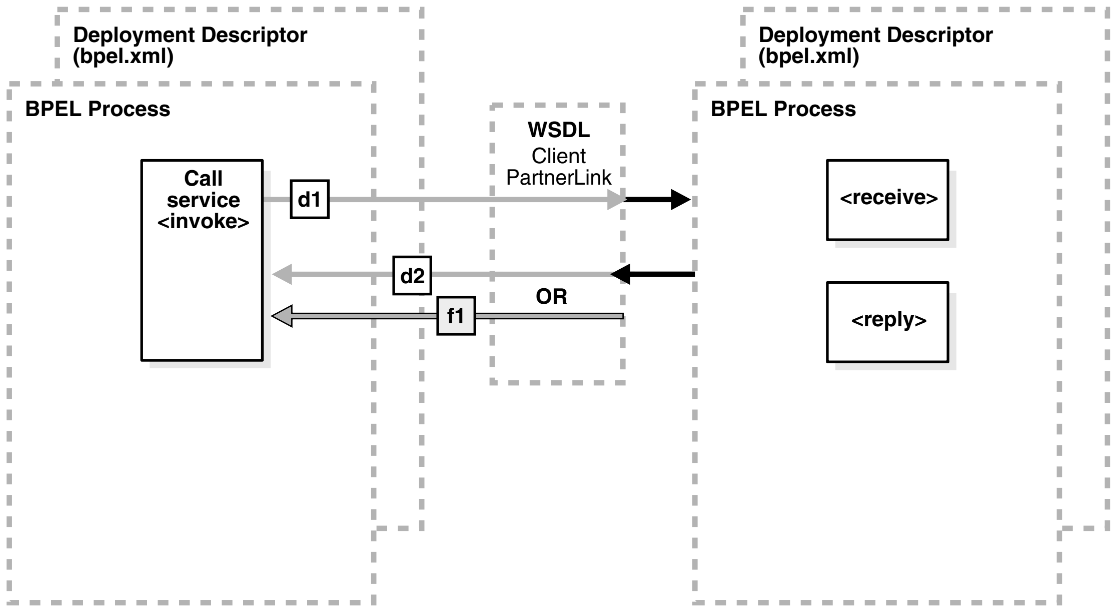
BPEL Process as the Service

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 differently depending on its role. Figure 13–2 provides an overview.

Figure 13–2 Synchronous 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.

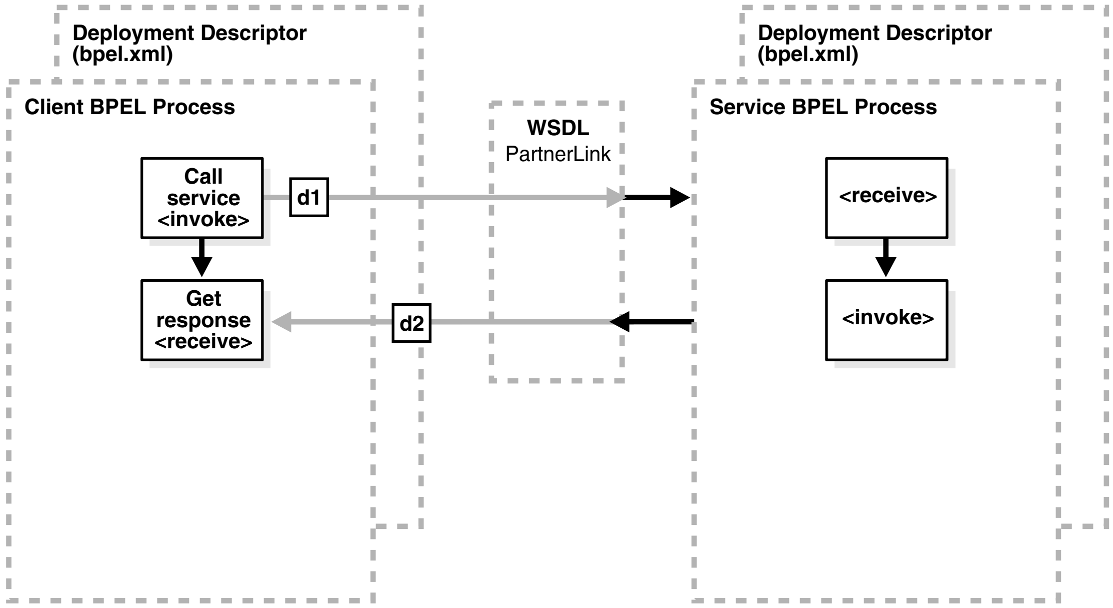
BPEL Process as the Service

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).

Asynchronous Interaction

In an asynchronous interaction, a client sends a request to a service and waits until the service replies. Figure 13–3 provides an overview.

Figure 13–3 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.

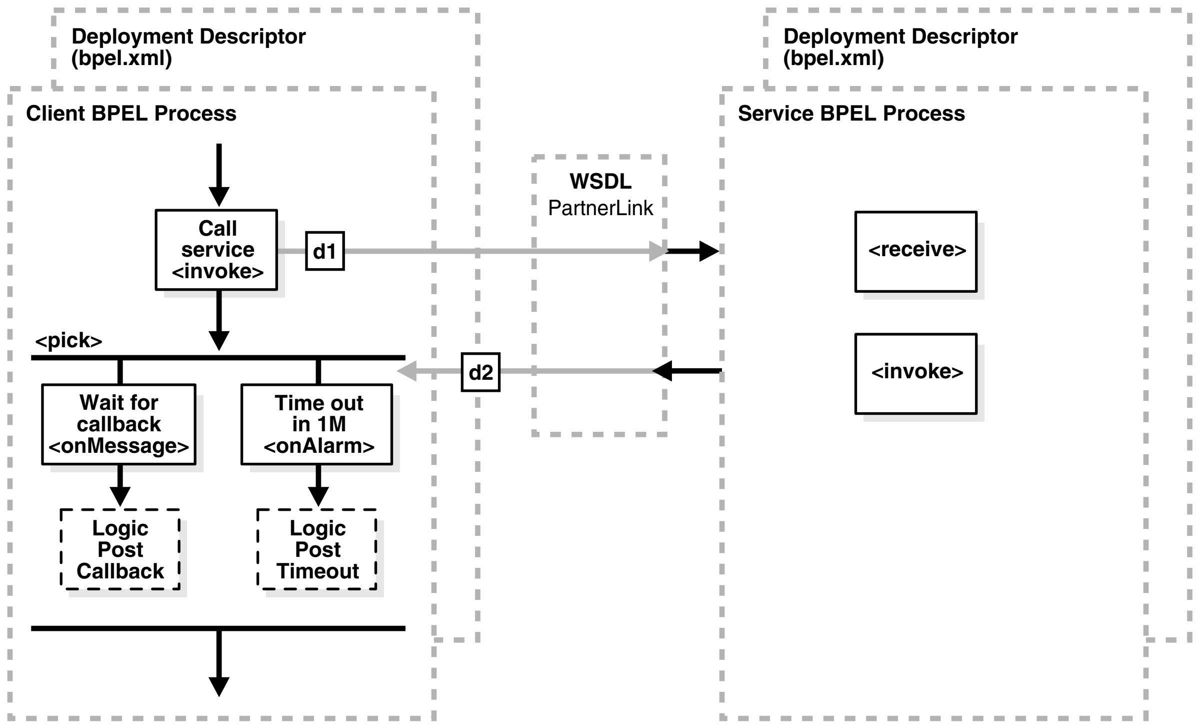
BPEL Process as the Service

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.

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 13–4 provides an overview.

Figure 13–4 Asynchronous Interaction with Timeout



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.

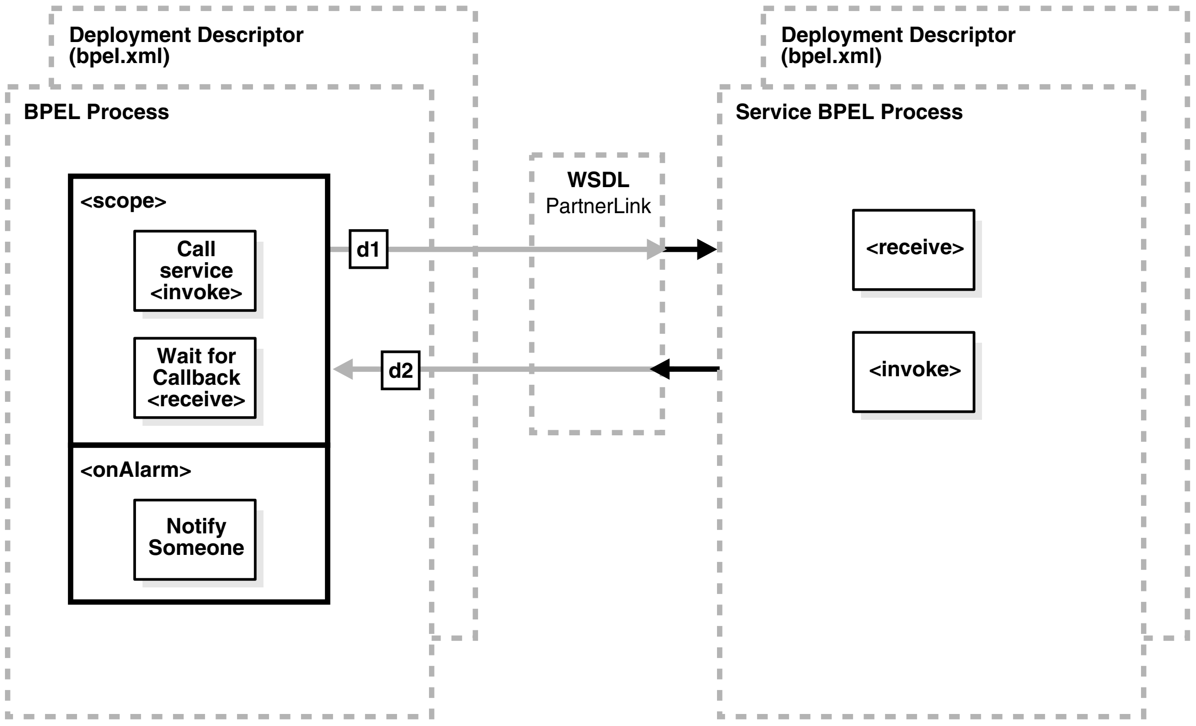
BPEL Process as the Service

The behavior is the same as with the asynchronous interaction with the BPEL Process as the service described in "BPEL Process as the Service" on page 13-4.

Asynchronous Interaction with a Notification Timer

In this interaction, 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 13–5 provides an overview.

Figure 13–5 Asynchronous Interaction with a Notification Time



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 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.

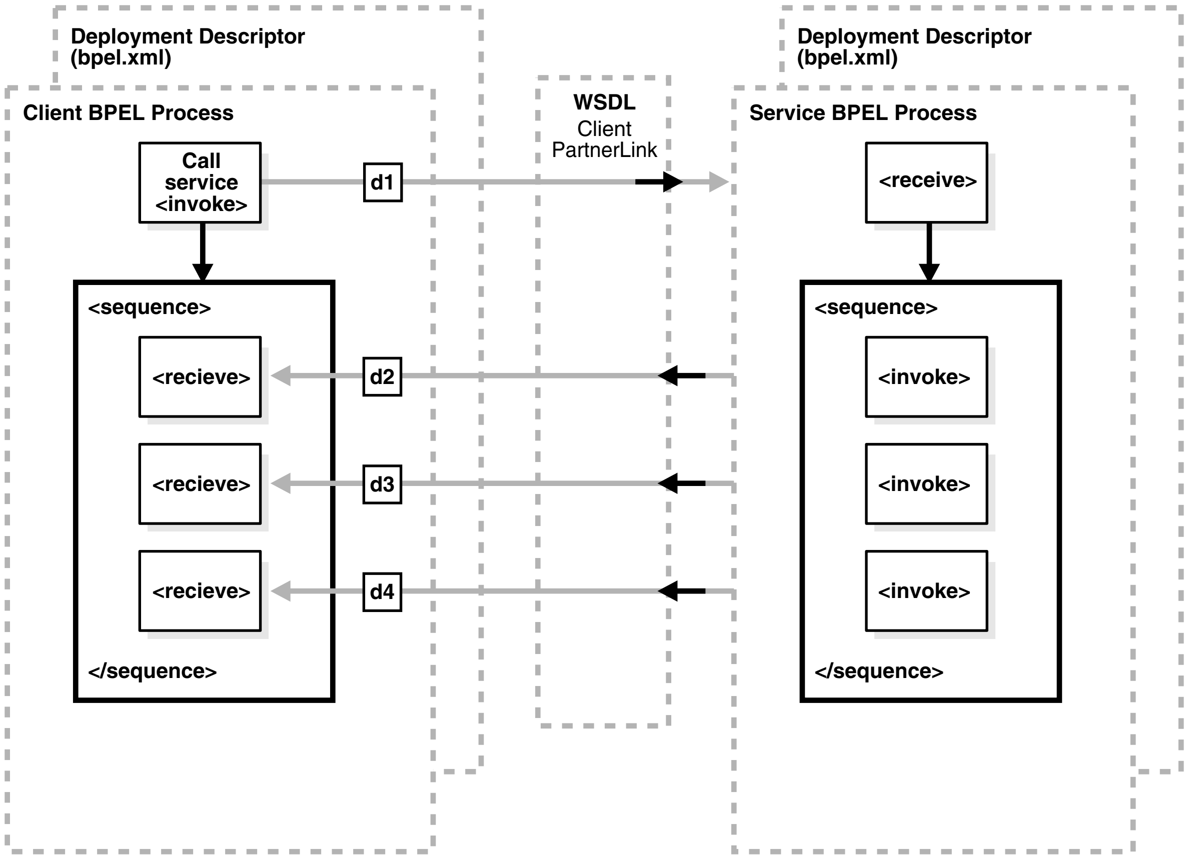
BPEL Process as the Service

The behavior is the same as with the asynchronous interaction with the BPEL process as the service described in "BPEL Process as the Service" on page 13-4.

One Request, Multiple Responses

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. Note that in this example, the number and types of responses are known. Figure 13–6 provides an overview.

Figure 13–6 One Request, Multiple Responses



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 attribute with three receive activities, one for each reply.

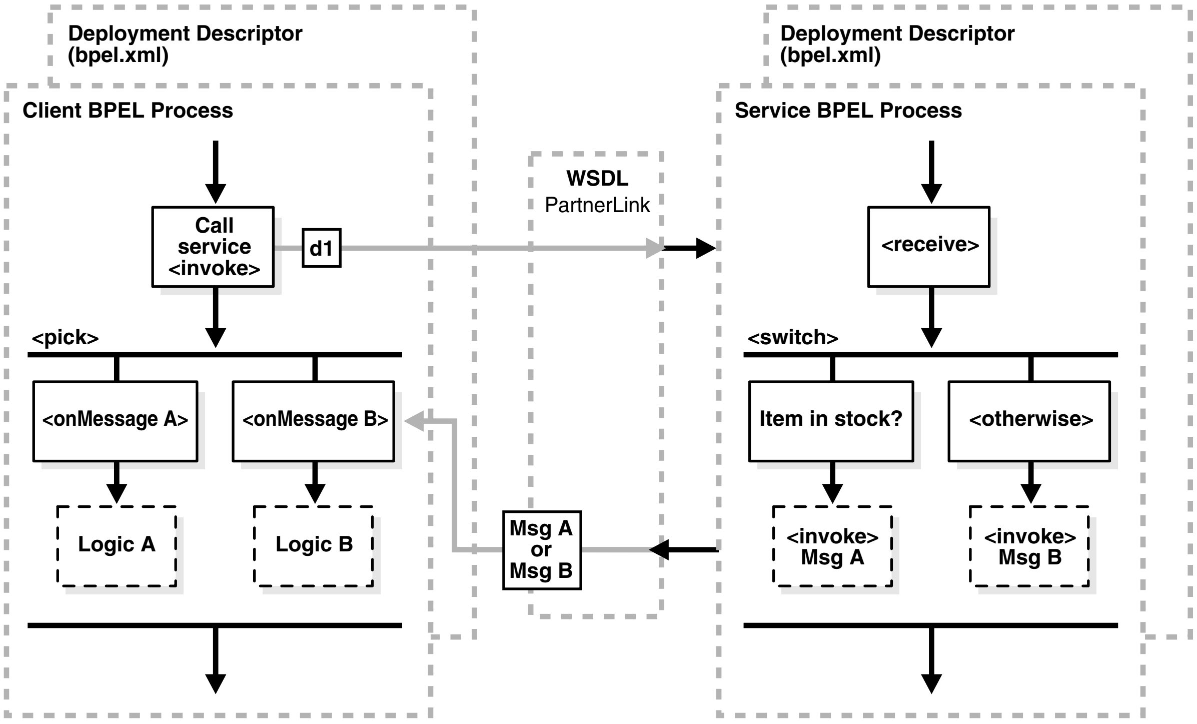
BPEL Process as the Service

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

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 13–7 provides an overview.

Figure 13–7 One Request, One of Two Possible Responses



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, 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, and another onMessage for the out-of-stock response and instructions on what to do if an out-of-stock message is received.

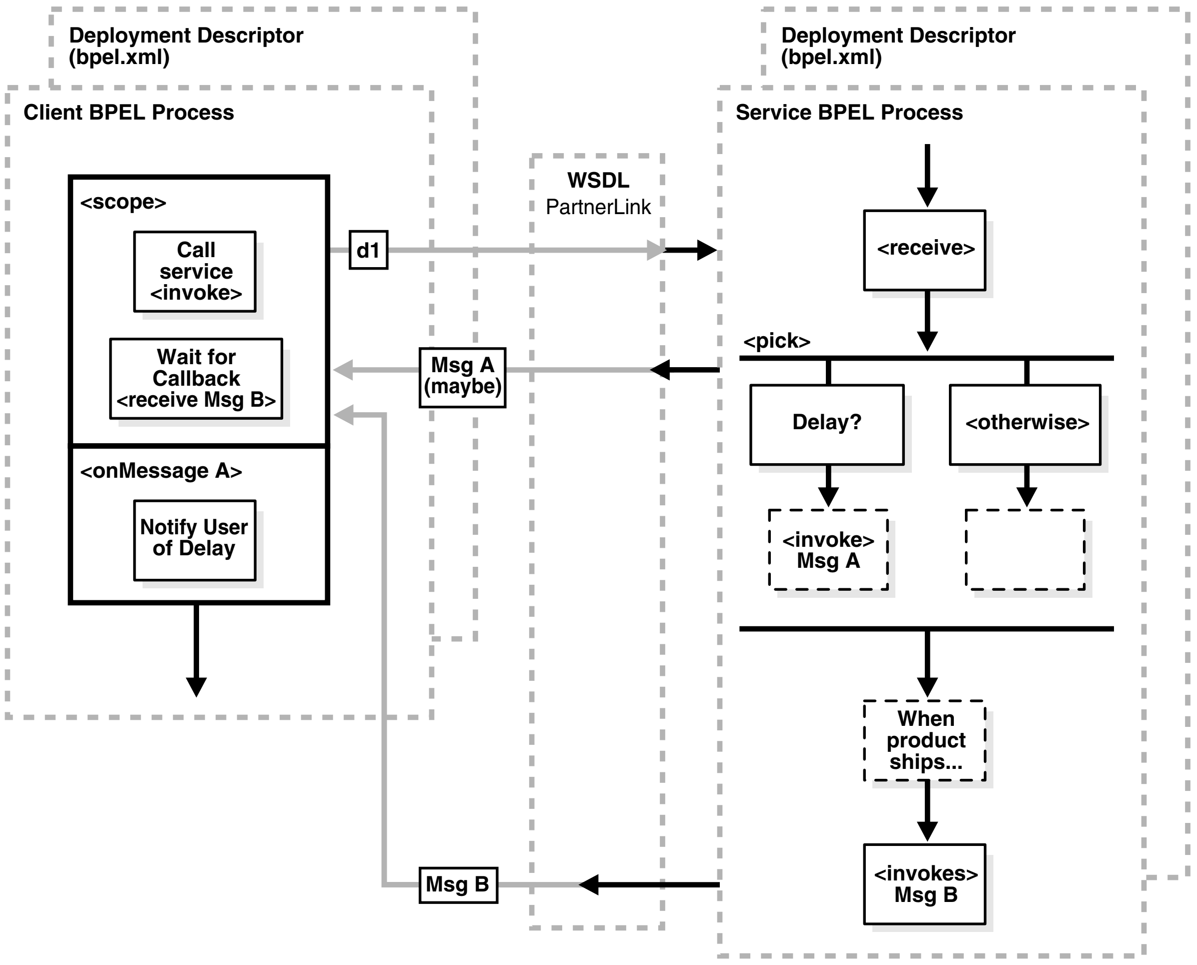
BPEL Process as the Service

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 another 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

The client sends a single request to a service and receives one or two responses. For example, the request can be to order a product online, and the service can send a delayed message if the product is delayed, and always sends a notification when the item ships. Figure 13–8 provides an overview.

Figure 13–8 One Request, a Mandatory Response, and an Optional Response



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 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.

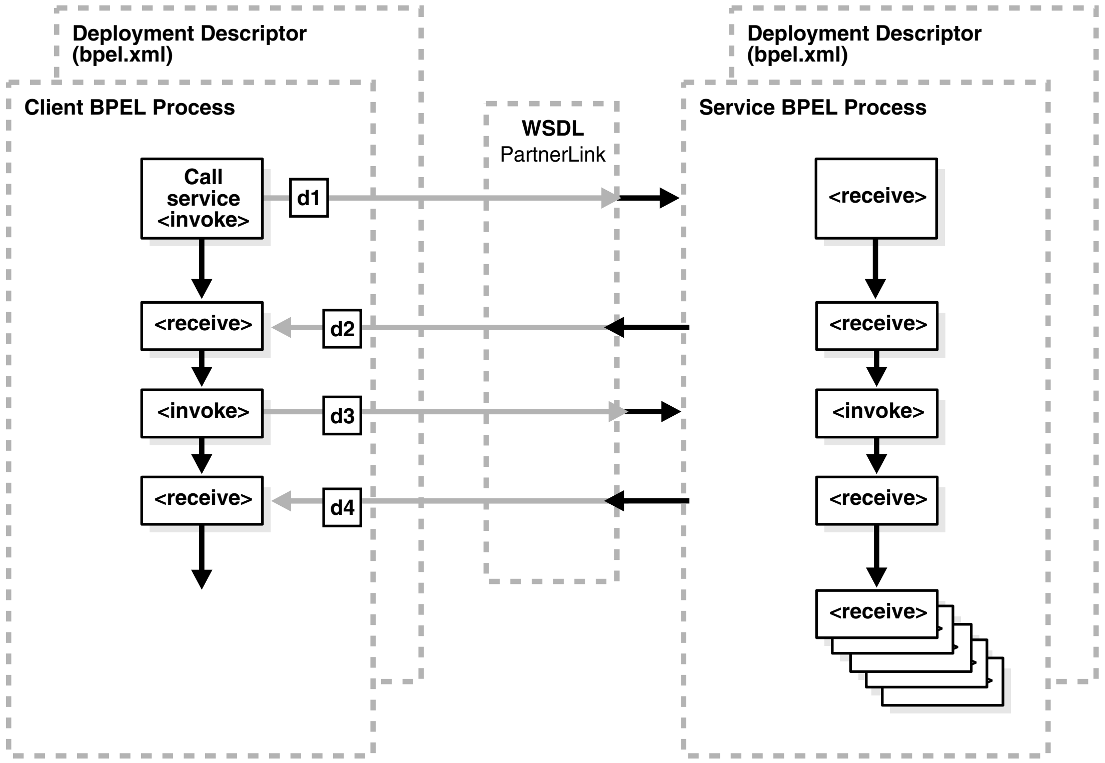
BPEL Process as the Service

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

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 13–9 provides an overview.

Figure 13–9 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.

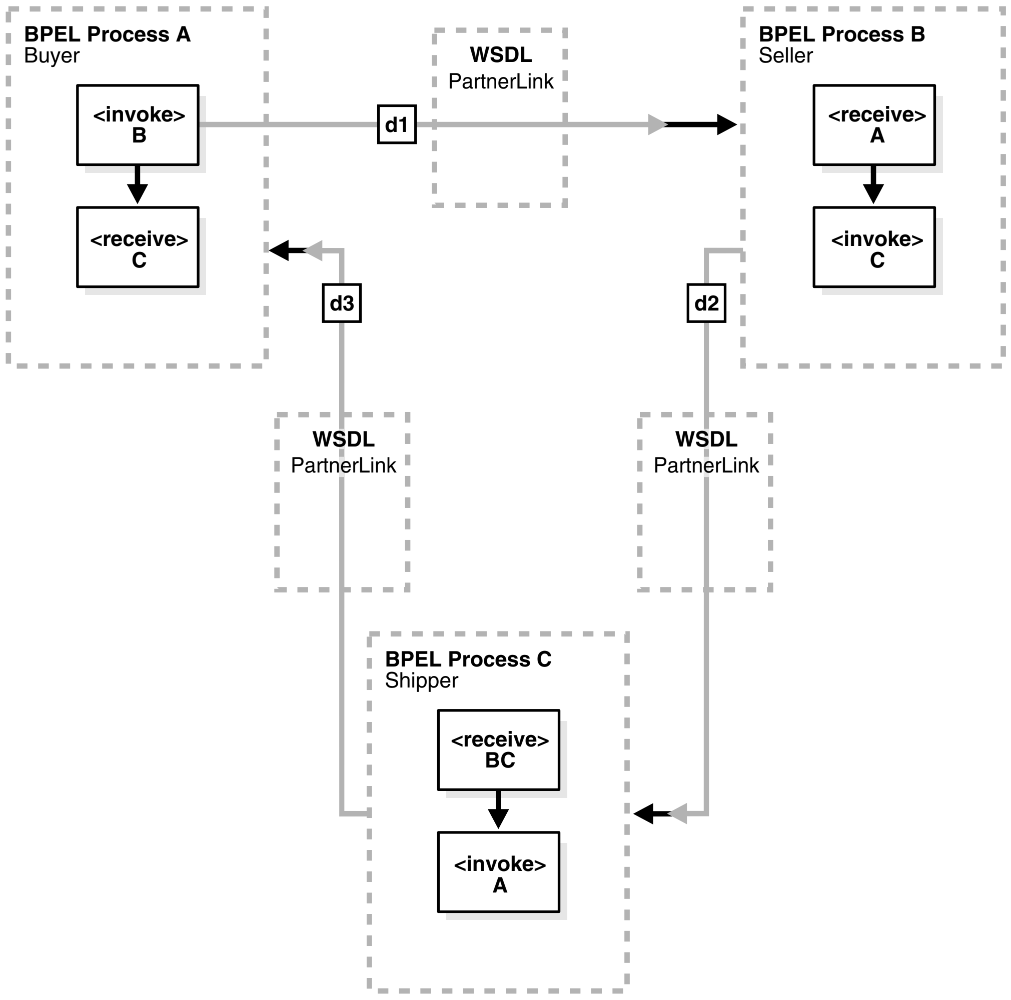
BPEL Process as the Service

The BPEL service needs a receive activity for each request, and a reply activity for each response. Once the responses are finished, the 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.

Third-Party 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 > B > C > A transaction pattern can also handle many transaction at once, and therefore needs a mechanism for keeping track of which message goes where. Figure 13–10 provides an overview.

Figure 13–10 Third-Party Interactions



See Also: "Correlations" on page 6-6 for more information about WS-Addressing and Correlation Sets

Summary

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.

Part III

Oracle BPEL Process Manager Services

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 14, "XSLT Mapper and Transformations"
- Chapter 15, "Oracle BPEL Process Manager Notification Service"
- Chapter 16, "Oracle BPEL Process Manager Workflow Services"
- Chapter 17, "Worklist Application"
- Chapter 18, "Sensors"

14 XSLT Mapper and Transformations

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 a Transform Activity
- The XSLT Mapper
- Step 1: Creating an XSL Map
- Step 2: Using the Mapper
- Step 3: Testing the Map
- Summary

Use Case for Transformation

Transformation use is demonstrated in the sample `XSLMapper`.

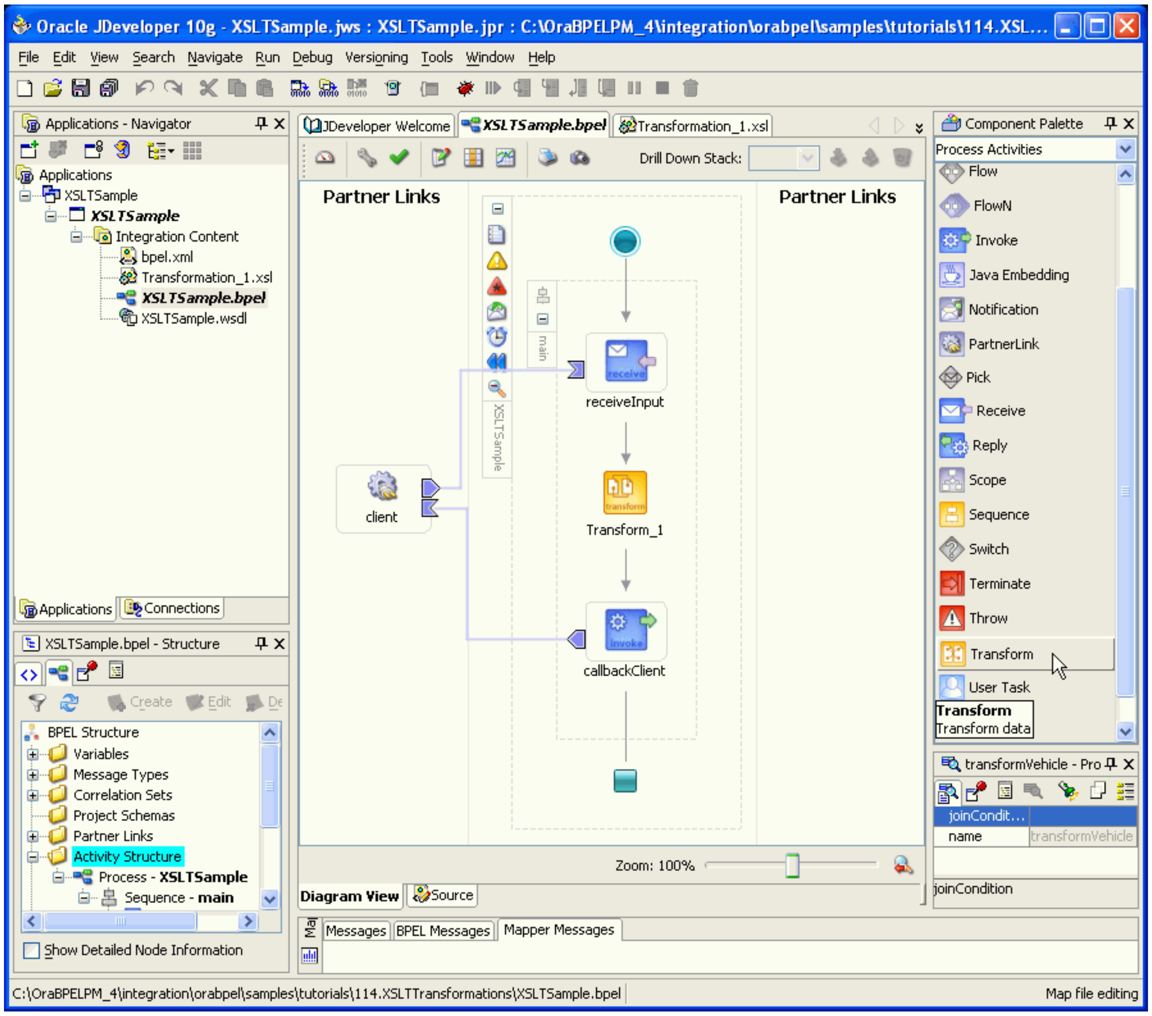
See:

- *`Oracle_home`*`\integration\orabpel\samples\demos\XSLMapper`
- *Oracle BPEL Process Manager Order Booking Tutorial*

Creating a Transform Activity

To create a transformation that maps a purchase order schema to an invoice schema, you drag and drop a **transform** activity from the **Component Palette** into your BPEL process diagram, as shown in Figure 14–1.

Figure 14–1 Invoking the Mapper from the Modeler

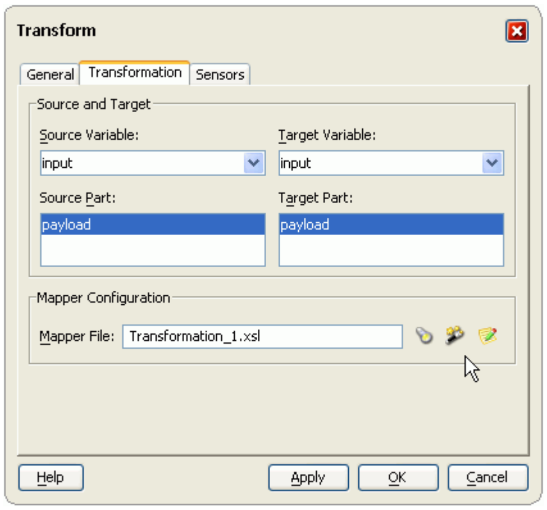


Double-click the **transform** activity and do the following. Figure 14–2 shows the Transform window.

- Select the source variable, source part, target variable, and target part.

- If the map file already exists, click the **flashlight** icon (first icon) to browse mappings.
- If the map file does not exist, enter a name for the map file in the **Mapper File** field and click the magic wand icon (second icon) to create a new mapping.
- If the map file already exists and you want to edit it, click the note pad icon (third icon) to edit the mapping.

Figure 14–2 Transform Window



Source and Target

Source Variable: input

Target Variable: input

Source Part: payload

Target Part: payload

Mapper Configuration

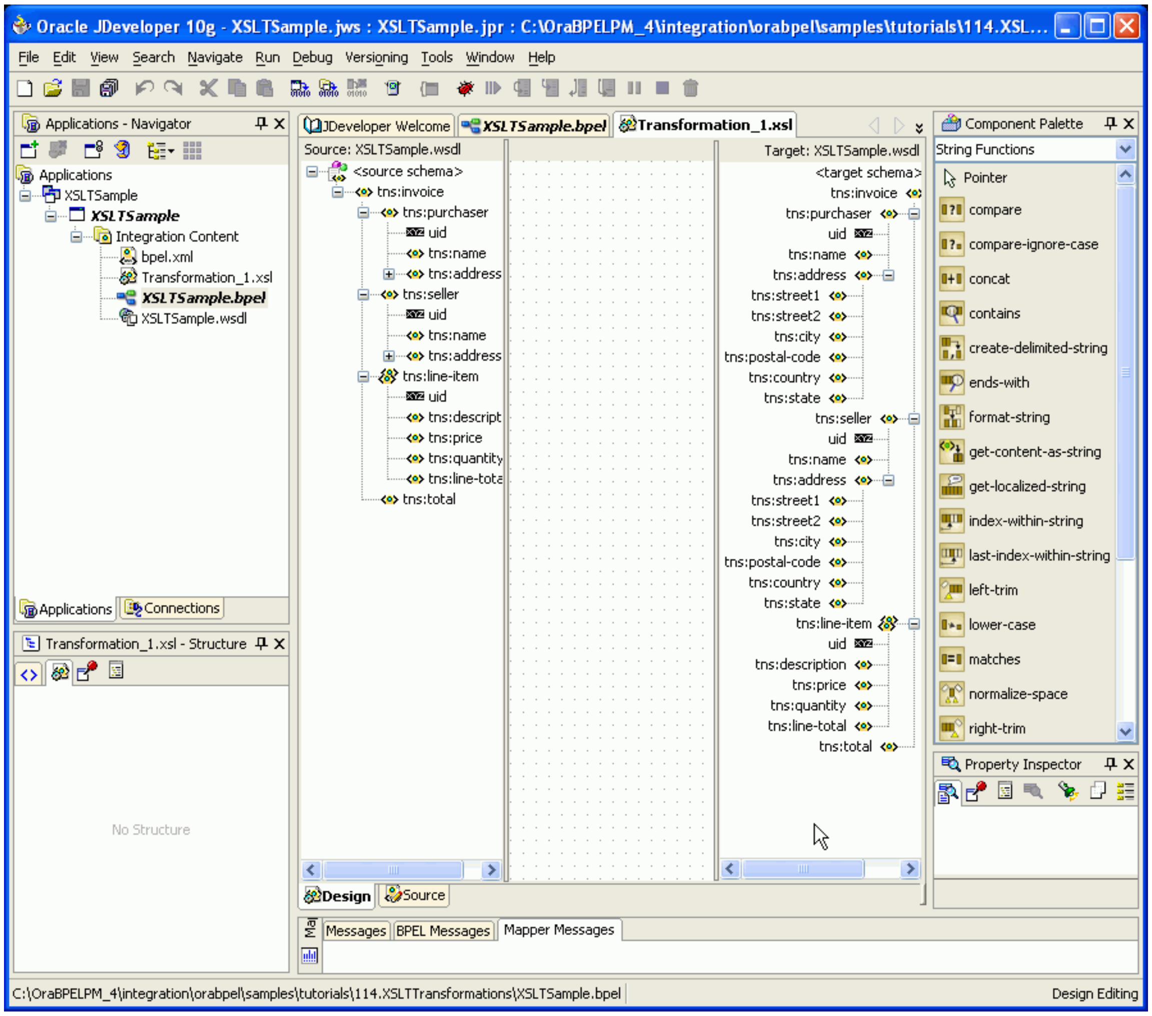
Mapper File: Transformation_1.xsl

See Also: The online Help for the Transform window for detailed descriptions of the fields

The XSLT Mapper

You use the XSLT Mapper transformation tool to create a map file. Figure 14–3 shows the layout of the XSLT Mapper.

Figure 14–3 Layout of the Mapper



The **Source** and the **Target** schemas are represented as trees and the nodes in the tree 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; 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** (for example, type, cardinality, and so on). The **Function Palette** is the container for all functions provided by the XSLT Mapper. The mapping 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
- One in the mapper pane or canvas

Right-click in each of the three separate panels to see what the context menus look like. A full set of **undo**, **redo**, and **delete** functions are also available in the main Edit and context menus.

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.

Step 1: Creating an XSL Map

The following steps provide a high level overview of how to create an XSL Map using `po.xsd` and `invoice.xsd` file in the `Oracle_home\integration\orabpel\samples\demos\XSLMapper` directory.

1. In JDeveloper BPEL Designer, select the workspace project in which you want to create the new XSL Map.
2. Add the **po.xsd** and **invoice.xsd** files to the project.
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.

   – 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. Under Source, expand **Project Schema Files** > **po.xsd**. Select **PurchaseOrder** as the root element for the source.
7. Under Target, expand **Project Schema files** > **invoice.xsd**. Select **Invoice** as the root element for the target.
8. Click **OK**.

   A new XSL Map is created.

Step 2: Using the Mapper

This section contains the following topics:

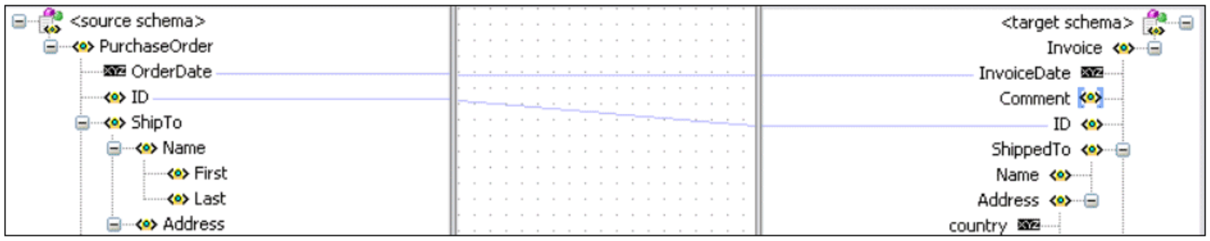
- Simple Copy by Linking Nodes
- Setting Constant Values
- Functions
- Editing XPath Expressions

- Adding XSLT Constructs
- Auto Mapping
- Generating Dictionaries

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 14–4.

Figure 14–4 Linking Nodes



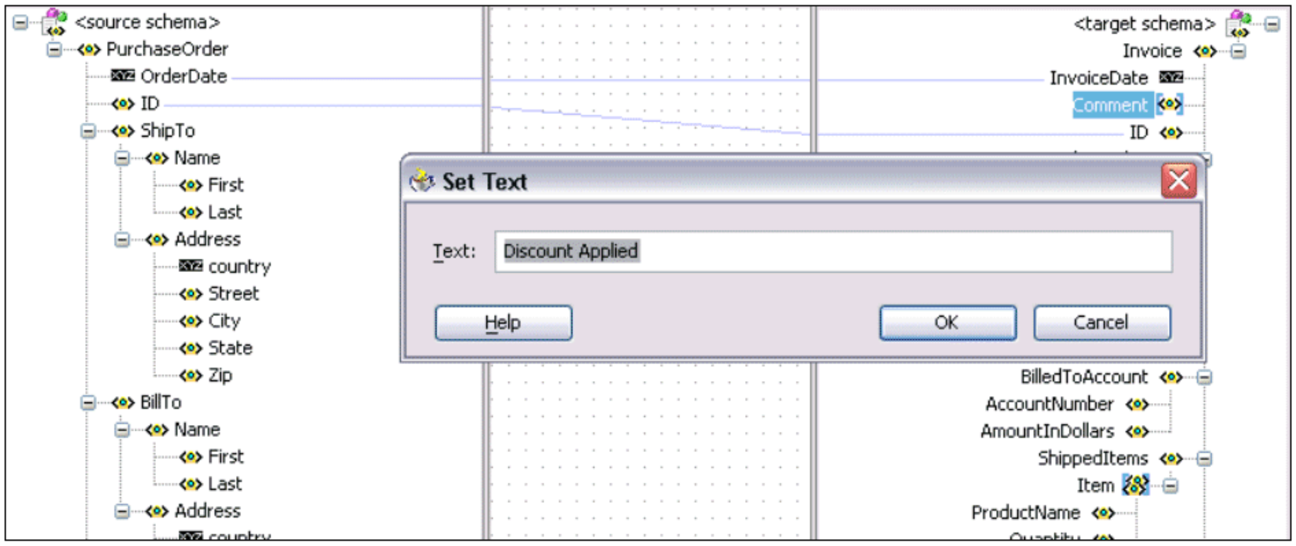
Setting Constant Values

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.
5. Click **OK** to save the text.

   A **T** icon is displayed next to a node that has text associated with it.
6. Set a constant value, **Discount Applied** to **Invoice/Comment**, as shown in Figure 14–5. To remove the text associated with a node, invoke the Set Text window again. Delete the contents and click **OK**.

Figure 14–5 Set Text Window



Text: Discount Applied

See Also: The online Help for the Set Text window for detailed information

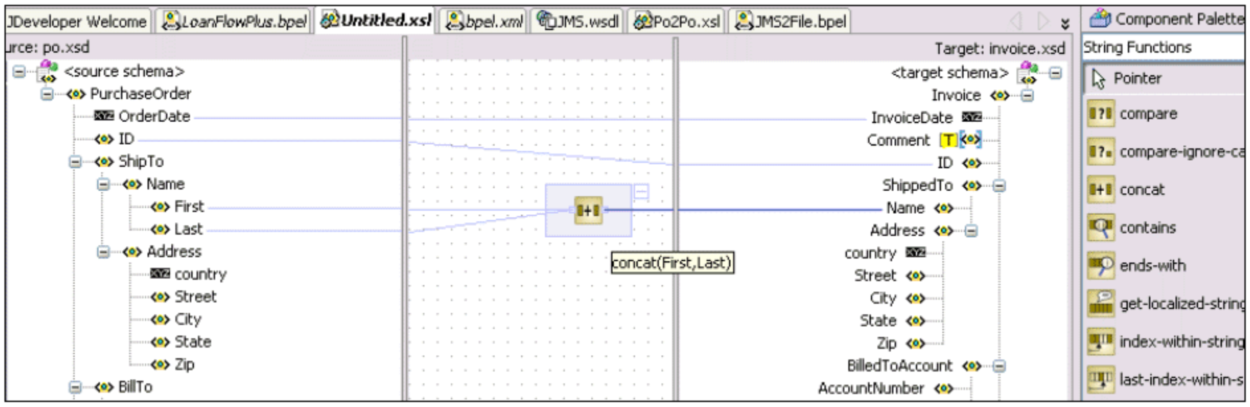
Functions

In addition to the standard XPath 1.0 functions, the Mapper provides a number of prebuilt extension functions (prefixed with `xp20` or `orcl`) 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 **Function Description**. A window with a description of the function appears.
4. Drag a **concat** function into the mapping 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 handle on the **concat** function, and also dragging a thread from the **ShippedTo** name and connecting it to the right handle on the **concat** function, as shown in Figure 14–6.

Figure 14–6 Using the Concat Function

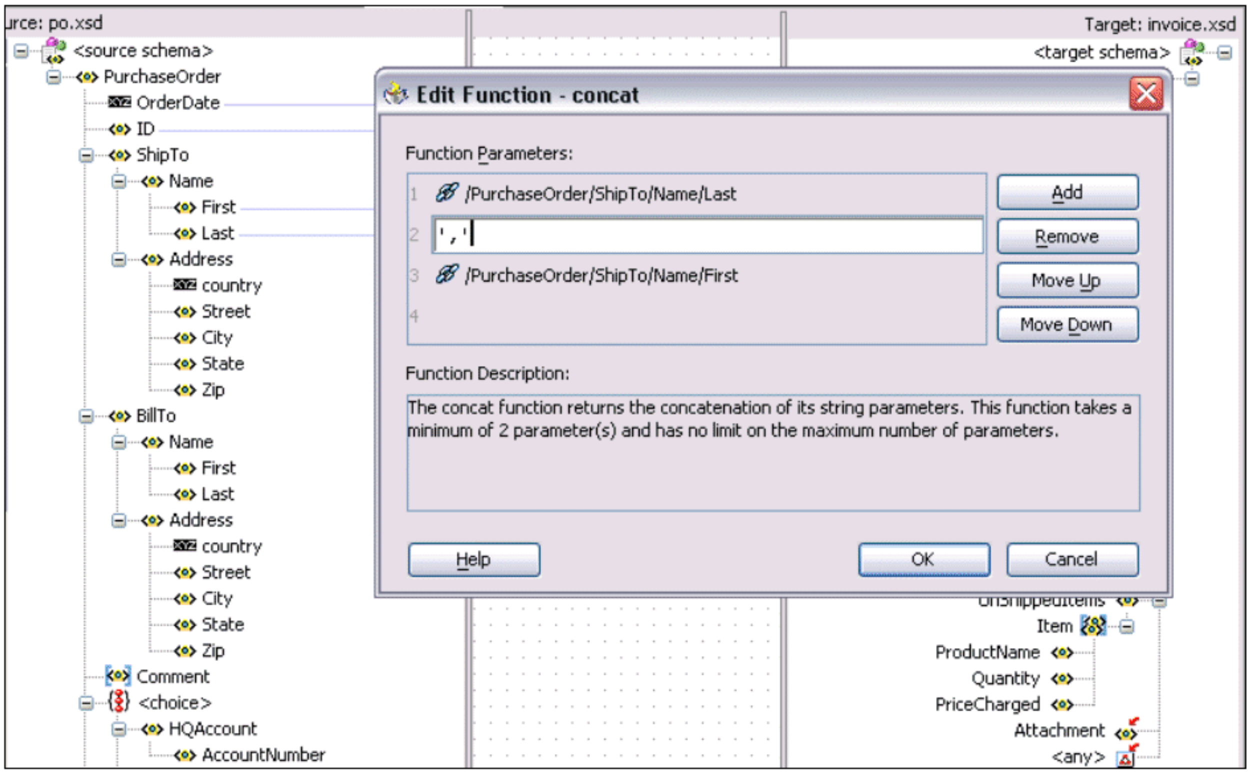


See Also: The documentation for the XPath extension functions, which is described in Appendix G, "XPath Extension Functions" and is also accessible by clicking **Manage BPEL Domain** > **XPath Library** in Oracle BPEL Console

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 the above output.

Figure 14–7 Editing Function Parameters



1 /PurchaseOrder/ShipTo/Name/Last
2 ','
3 /PurchaseOrder/ShipTo/Name/First
4

The concat function returns the concatenation of its string parameters. This function takes a minimum of 2 parameter(s) and has no limit on the maximum number of parameters.

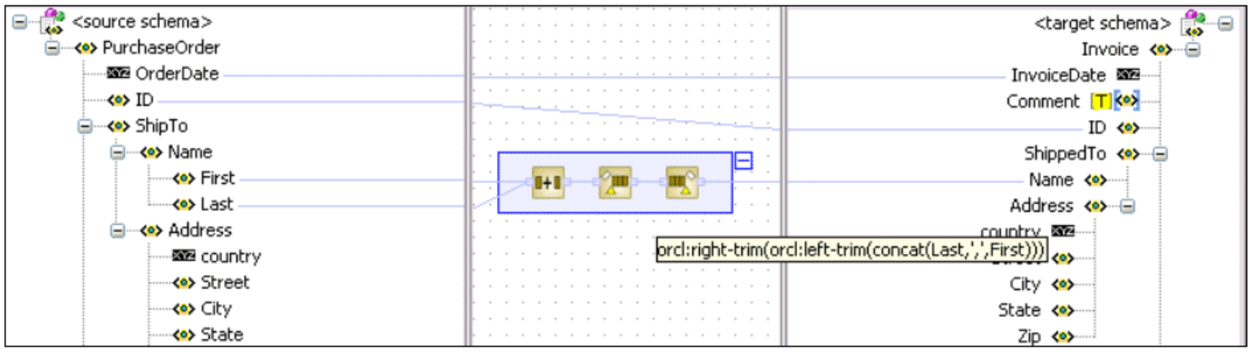
See Also: The online Help for the Edit Function window by clicking the **Help** button to see how to add, remove, and reorder function parameters

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 14–8.

The chaining function can also be defined by dragging and dropping the function to a connecting link.

Figure 14–8 Chaining Functions



orcl:right-trim(orcl:left-trim(concat(Last,',',First)))

Named Templates and User-Defined Functions

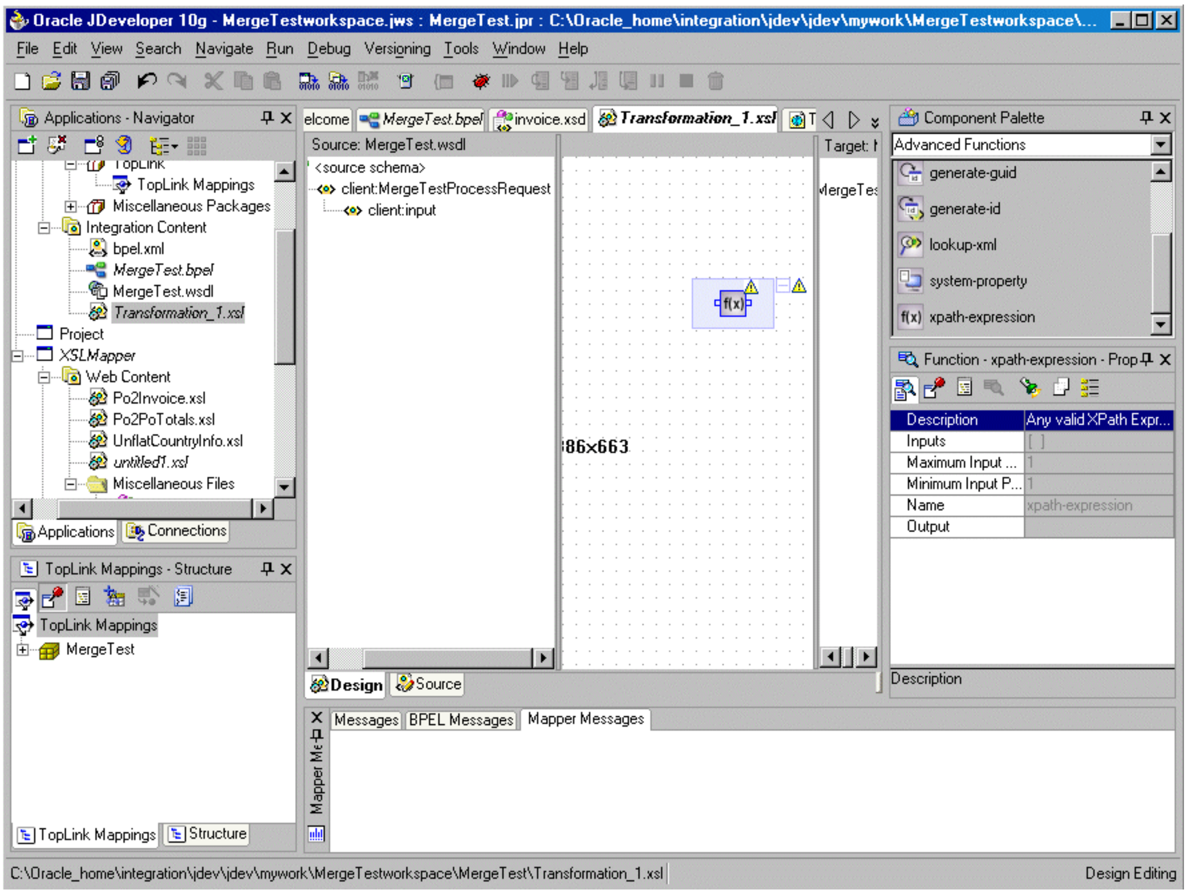
To use named templates in the design view, you code the template in the source view. The template is now available in the function palette under the **Template** category and can be used like any other function. You can plug in your own set of Java functions, which appear in the function palette under the **User defined Extension Functions** category. They can be used like any other function.

See Also: `*Oracle_home*\integration\orabpel\samples\demos\XSLMapper\ExtensionFunctions\README.txt` for more information

Editing XPath Expressions

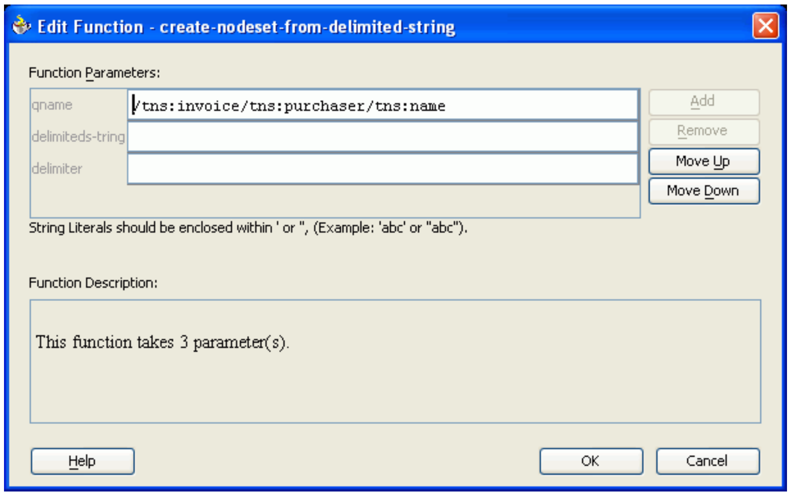
To use an XPath expression in a transformation mapping, select **Advanced Functions** from the **Component Palette**, and then drag and drop **xpath-expression** from the list into the transformation window, as shown in Figure 14–9.

Figure 14–9 Editing XPath Expressions



When you double-click the icon, the Edit Function window appears, as shown in Figure 14–10. You can press the **Ctrl** key and then the **spacebar** to invoke the XPath Building Assistant.

Figure 14–10 Edit Function Window


Edit Function - create-nodeset-from-delimited-string

Function Parameters:

qname	/tns:invoice/tns:purchaser/tns:name
delimiteds-tring	
delimiter	

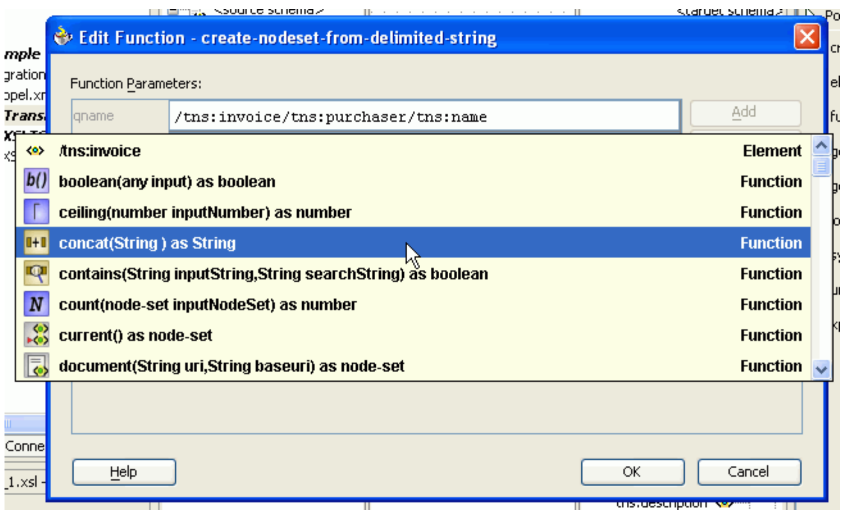
String Literals should be enclosed within ' or ", (Example: 'abc' or "abc").

Function Description:

This function takes 3 parameter(s).

Figure 14–11 shows the XPath Building Assistant.

Figure 14–11 The XPath Building Assistant



/tns:invoice/tns:purchaser/tns:name

/tns:invoice	Element
boolean(any input) as boolean	Function
ceiling(number inputNumber) as number	Function
concat(String) as String	Function
contains(String inputString,String searchString) as boolean	Function
count(node-set inputNodeSet) as number	Function
current() as node-set	Function
document(String uri,String baseuri) as node-set	Function

See Also: The online Help for the Edit Function window, which includes a link to instructions on using 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 to bring up the context menu and choose the required XSLT element in the menu.

See Also: *Oracle BPEL Process Manager Order Booking Tutorial* for an example of a for-each node

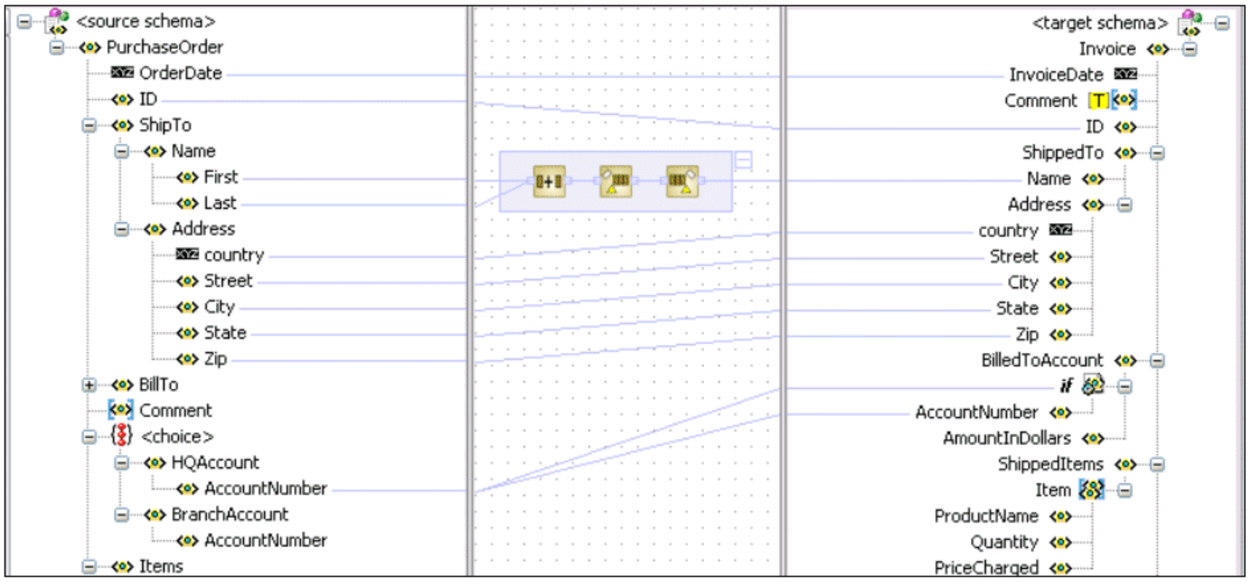
Conditional Processing with xsl:if

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** > **if** and connect **PurchaseOrder/HQAccount/AccountNumber** to **Invoice/BilledToAccount/if**.
3. Connect **PurchaseOrder/HQAccount/AccountNumber** to **Invoice/BilledToAccount/if/AccountNumber**.

Figure 14–12 shows the results.

Figure 14–12 Conditional Processing with xsl:if



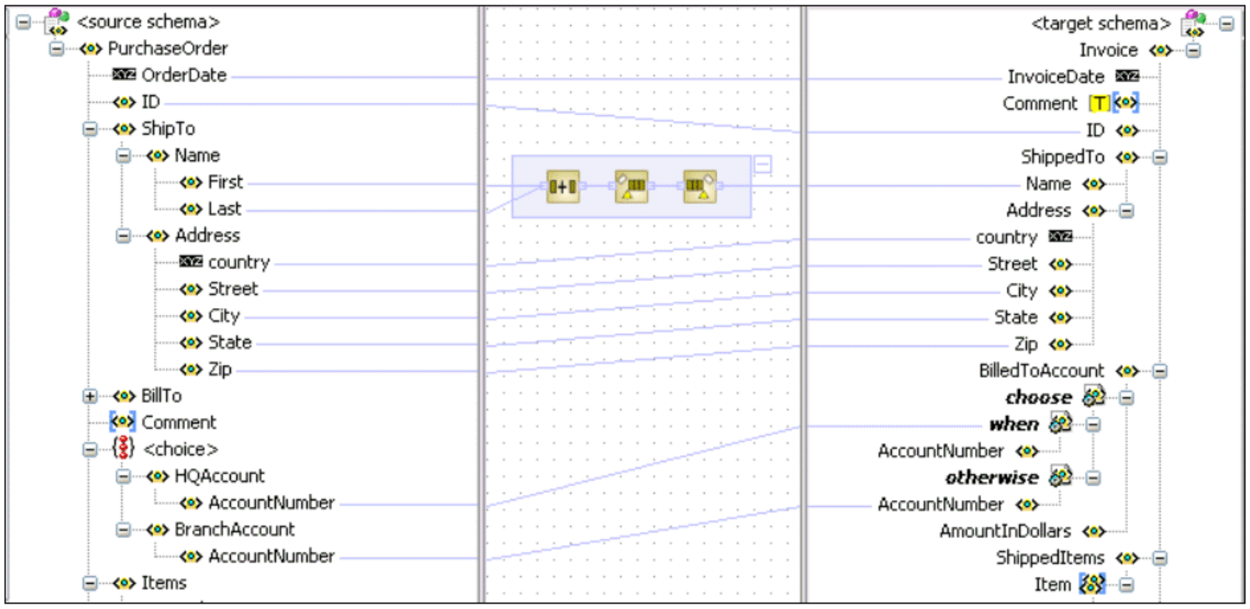
Conditional Processing with xsl:choose

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** > **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** > **choose** in the target tree and right-click to bring up the context sensitive menu.
5. Select **Insert XSL node** > **otherwise** from the menu.
6. Connect **PurchaseOrder/BranchAccount/AccountNumber** to **Invoice/BilledToAccount/choose/otherwise/AccountNumber**.

Figure 14–13 shows the results.

Figure 14–13 Conditional Processing with xsl:choose



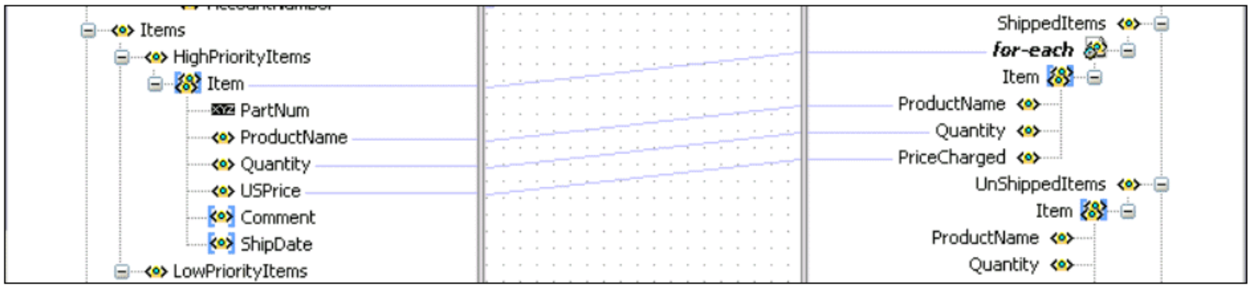
Handling Repetition or Arrays

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** > **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 14–14 shows the results.

Figure 14–14 Handling Repetition or Arrays

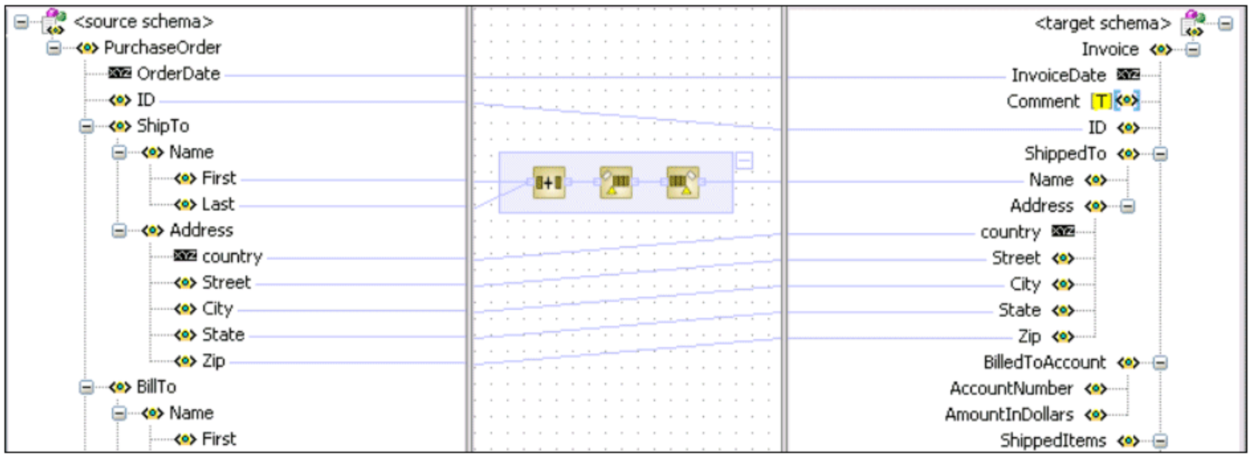


Note: Executing an auto map automatically inserts **xsl:for-each.** To see the auto map in use, drag and drop **PurchaseOrder/Items/LowPriorityItems** to **Invoice/UnShippedItems**; **for-each** is automatically created.

Auto Mapping

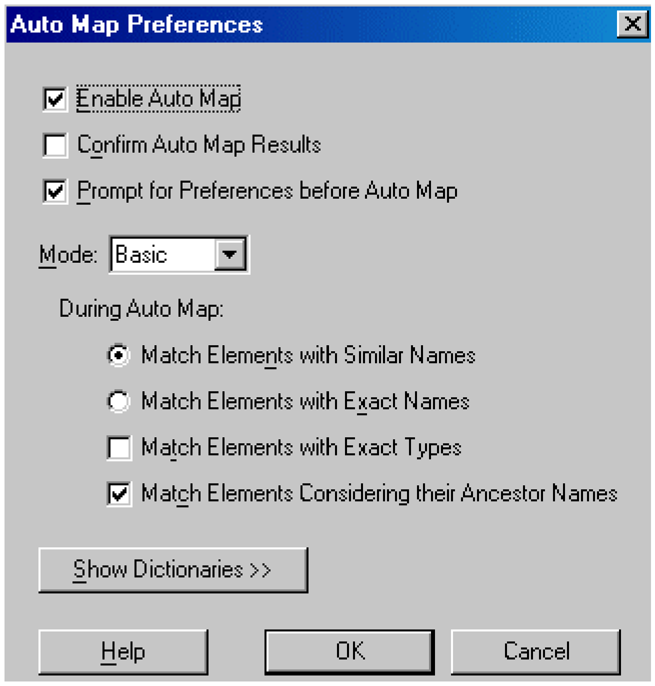
Mapping nonleaf nodes starts the auto map feature. The system automatically tries to link all relevant nodes under the selected source and target. Try the auto map feature by mapping **PurchaseOrder > ShipTo > Address** to **Invoice > ShippedTo > Address**. All nodes under **Address** are automatically mapped, as shown in Figure 14–15.

Figure 14–15 Auto Mapping



The behavior of the auto map can be tuned by altering the settings in JDeveloper BPEL Designer preferences or by right-clicking the transformation window and selecting **Auto Map Preferences**. This displays the window shown in Figure 14–16.

Figure 14–16 Auto Map Preferences


Auto Map Preferences

[✓] Enable Auto Map
[ ] Confirm Auto Map Results
[✓] Prompt for Preferences before Auto Map

Mode: Basic

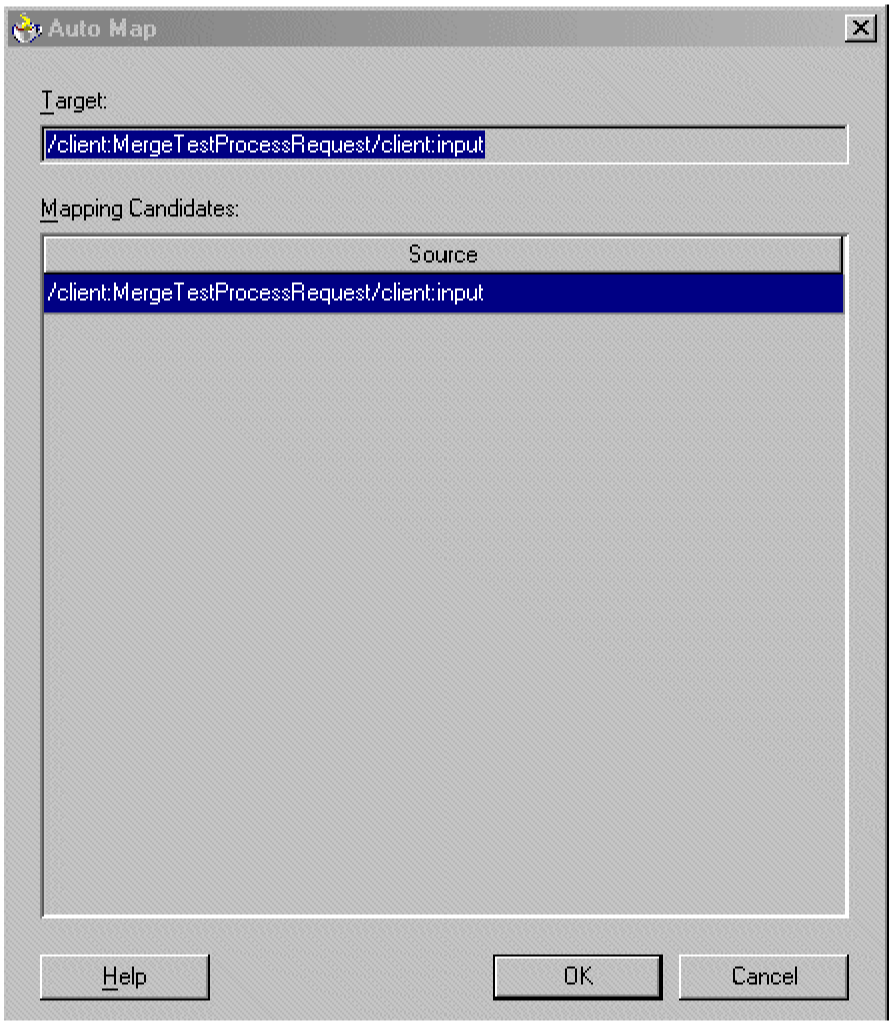
During Auto Map:

(●) Match Elements with Similar Names
( ) Match Elements with Exact Names
[ ] Match Elements with Exact Types
[✓] Match Elements Considering their Ancestor Names

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 14–17.

Figure 14–17 Auto Mapping Candidates

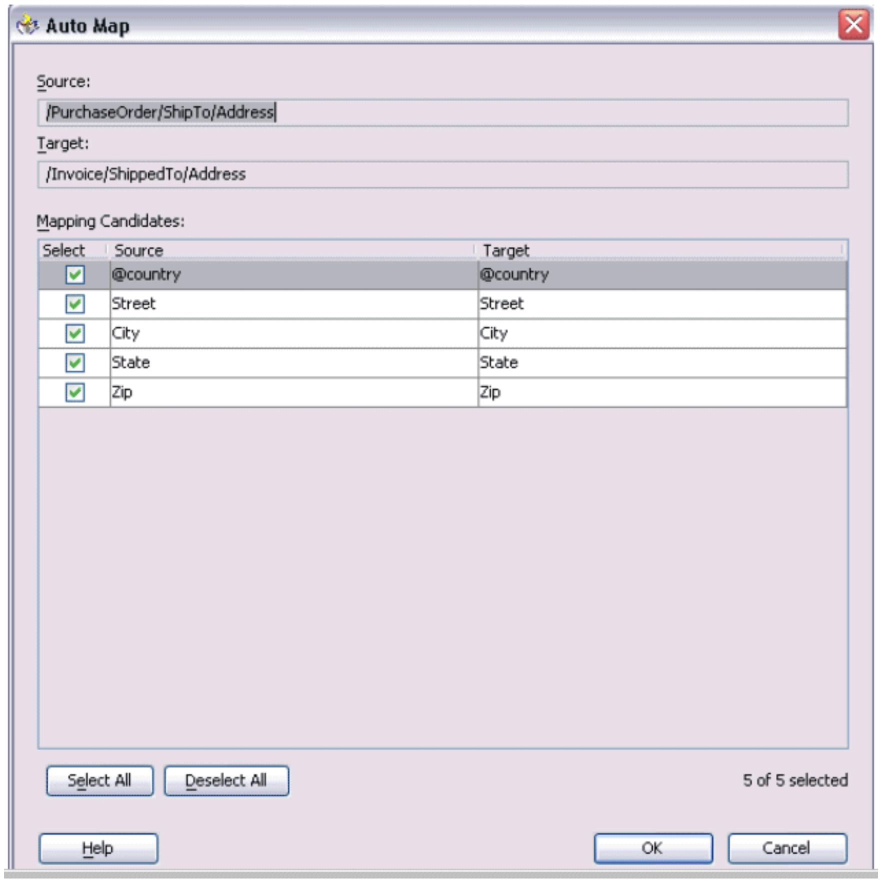

Auto Map
Target:
/client:MergeTestProcessRequest/client:input
Mapping Candidates:
Source
/client:MergeTestProcessRequest/client:input
Help
OK
Cancel

See Also: The online Help for the Auto Map window by clicking the **Help** button to see a description of the fields

Auto Map with Confirmation

When the **Confirm Auto Map Results** preference is set to on, 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 14–18. The window enables you to filter one or more mappings.

Figure 14–18 Auto Map with Confirmation



Auto Map

Source:

/PurchaseOrder/ShipTo/Address

Target:

/Invoice/ShippedTo/Address

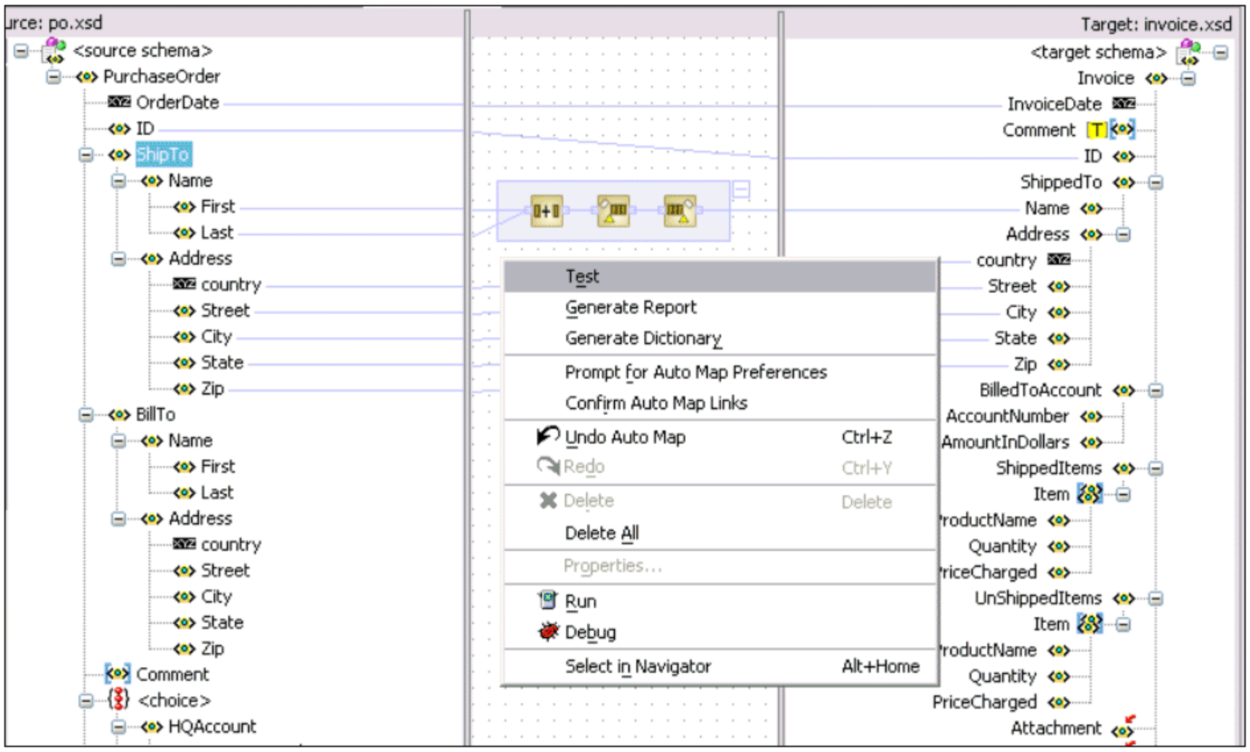
Mapping Candidates:

Select	Source	Target
✓	@country	@country
✓	Street	Street
✓	City	City
✓	State	State
✓	Zip	Zip

See Also: The online Help for the Auto Map window by clicking the **Help** button to see a description of the fields

The auto map confirmation window can be also be turned on or off with **Confirm Auto Map Links** in the context menu of the mapper pane, as shown in Figure 14–19.

Figure 14–19 Confirm Auto Map Links Selection



Generating Dictionaries

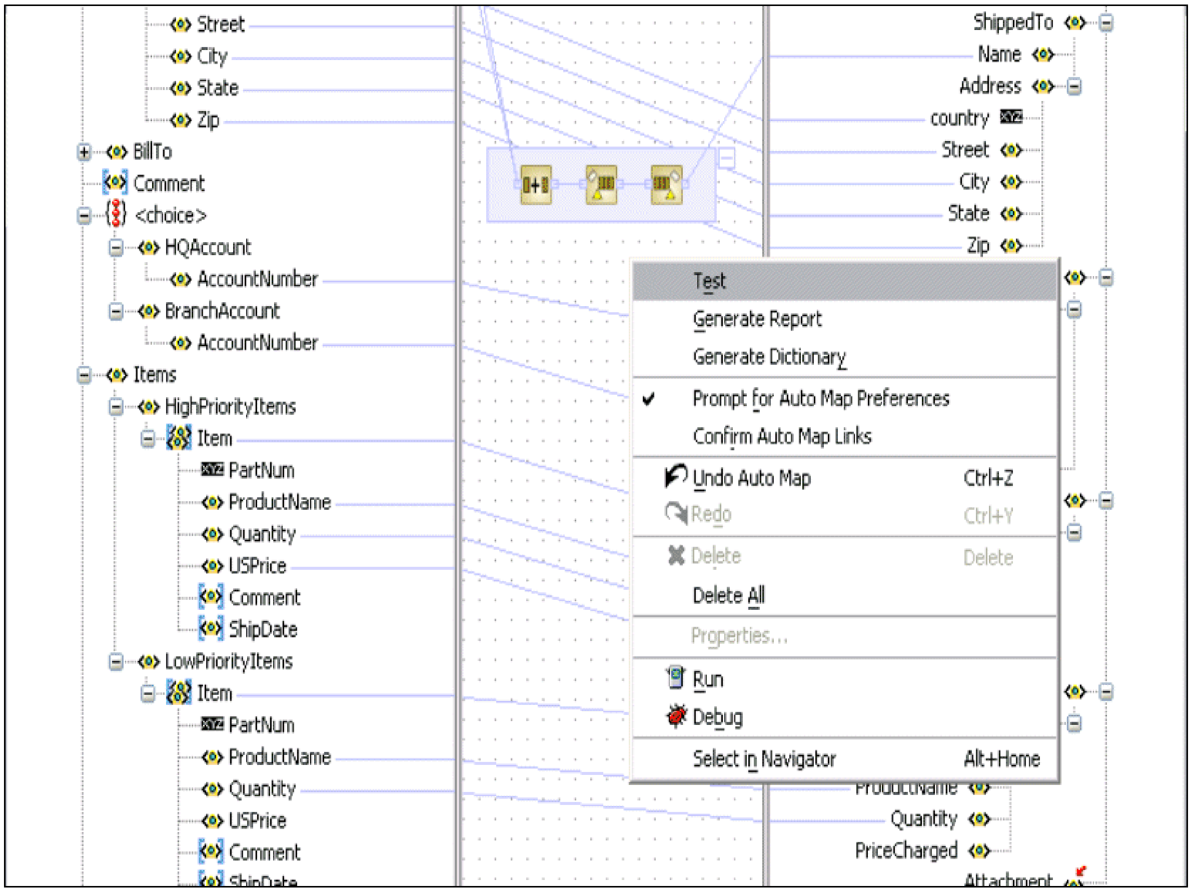
A dictionary is an XML file that captures the synonyms for mappings. Right-click the mapper pane as shown in Figure 14–19 and select **Generate Dictionary**. This prompts you for the dictionary name and the directory in which to place the dictionary. For example, you may want to map a purchase order 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 JDeveloper BPEL Designer.
5. Perform an automatic mapping from the purchase order to the purchase order acknowledgment.

Step 3: 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 pane context sensitive menu, as shown in Figure 14–20.

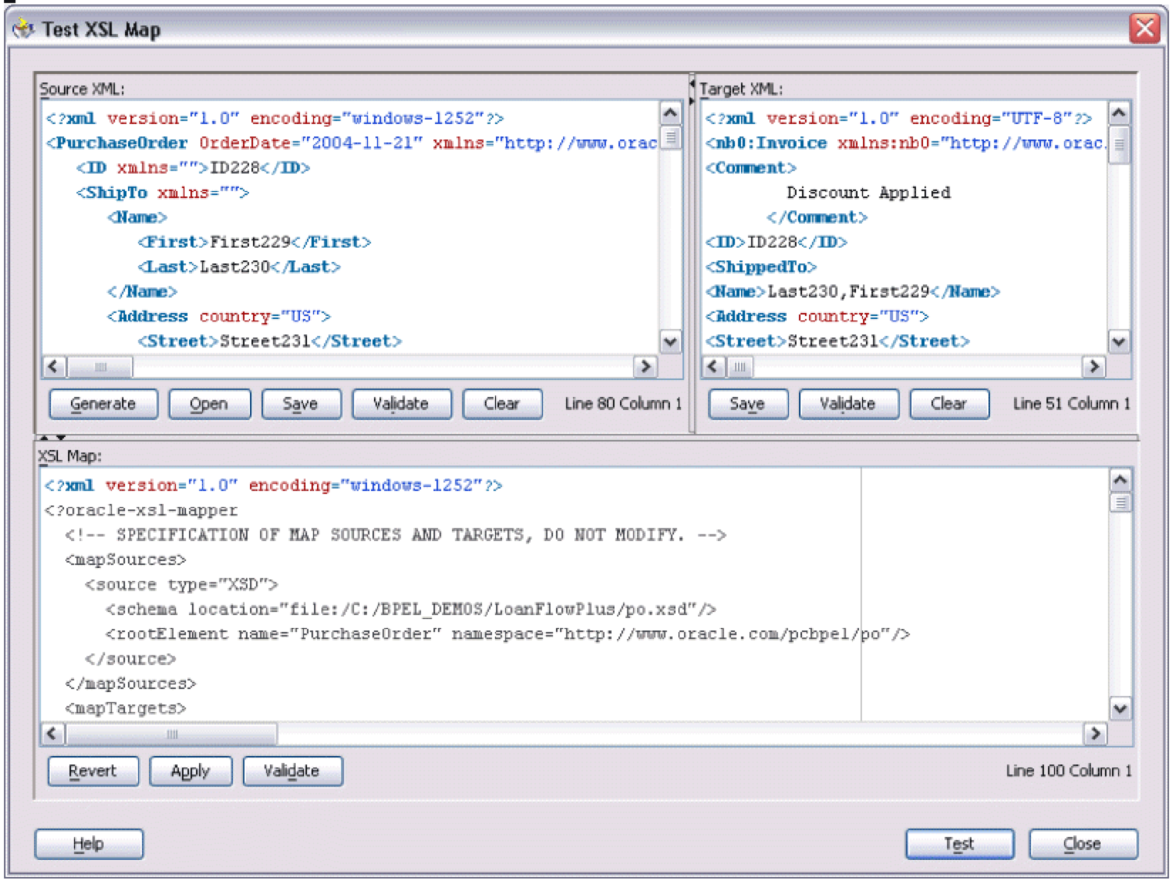
Figure 14–20 Invoking the Test Window



Test Window

The **Open** button is used to load a sample XML instance file for testing. If you do not have a source file or if you prefer the system to generate a random source instance, you can click the **Generate** button. After you have a sample source instance, you can generate the result document by clicking the **Test**, as shown in Figure 14–21.

Figure 14–21 Test Window



Source XML:

```
<?xml version="1.0" encoding="windows-1252"?>
<PurchaseOrder OrderDate="2004-11-21" xmlns="http://www.orac
   <ID xmlns="">ID228</ID>
   <ShipTo xmlns="">
      <Name>
         <First>First229</First>
         <Last>Last230</Last>
      </Name>
      <Address country="US">
         <Street>Street231</Street>
```

Target XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<nb0:Invoice xmlns:nb0="http://www.orac
<Comment>
        Discount Applied
      </Comment>
<ID>ID228</ID>
<ShippedTo>
<Name>Last230,First229</Name>
<Address country="US">
<Street>Street231</Street>
```

XSL Map:

```
<?xml version="1.0" encoding="windows-1252"?>
<?oracle-xsl-mapper
  <!-- SPECIFICATION OF MAP SOURCES AND TARGETS, DO NOT MODIFY. -->
  <mapSources>
    <source type="XSD">
      <schema location="file:/C:/BPEL_DEMOS/LoanFlowPlus/po.xsd"/>
      <rootElement name="PurchaseOrder" namespace="http://www.oracle.com/pcbpel/po"/>
    </source>
  </mapSources>
  <mapTargets>
```

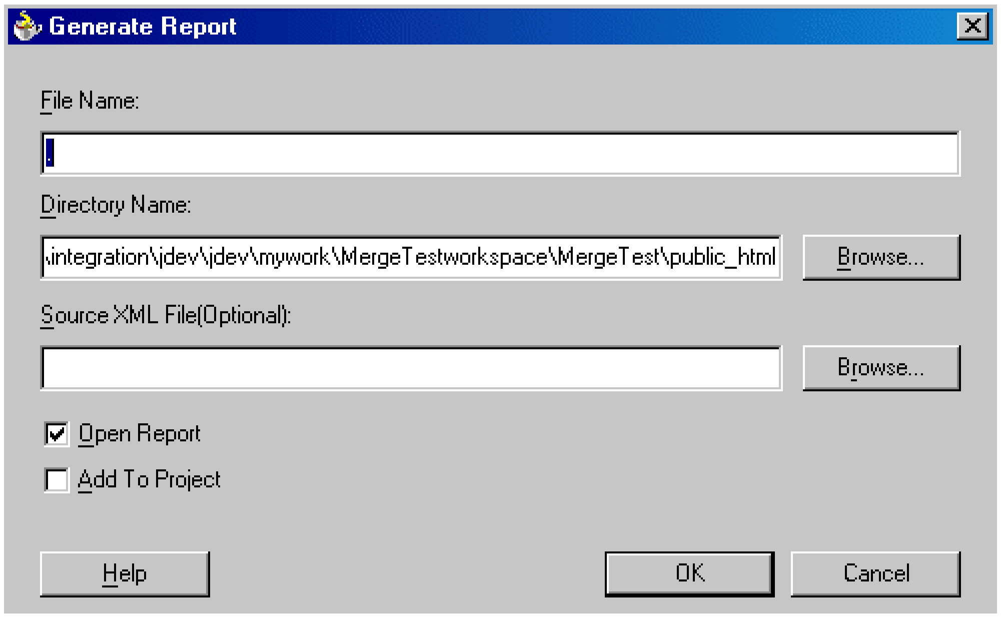
Generating Reports

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 14–22.

Figure 14–22 The Generate Report Window


Generate Report
File Name:
Directory Name:
\integration\jdev\jdev\mywork\MergeTestworkspace\MergeTest\public_html
Browse...
Source XML File(Optional):
Browse...
Open Report
Add To Project
Help
OK
Cancel

See Also: The online Help for the Generate Report window by clicking the **Help** button to see detailed information

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
- Automatically mapping target and source nodes
- Generating dictionaries
- 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.

15 Oracle BPEL Process Manager Notification Service

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, or short message service (SMS).

This chapter contains the following topics:

- Use Cases for Notification Service
- Overview of Notification Service Concepts
- Configuring Notification Service in JDeveloper BPEL Designer
- 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, JDeveloper BPEL Designer uses the notification service to alert users by voice 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 JDeveloper BPEL Designer and send an e-mail with an attachment.

> **See:** `*Oracle_Home*\integration\orabpel\samples\tutorials\`

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, or an SMS message.

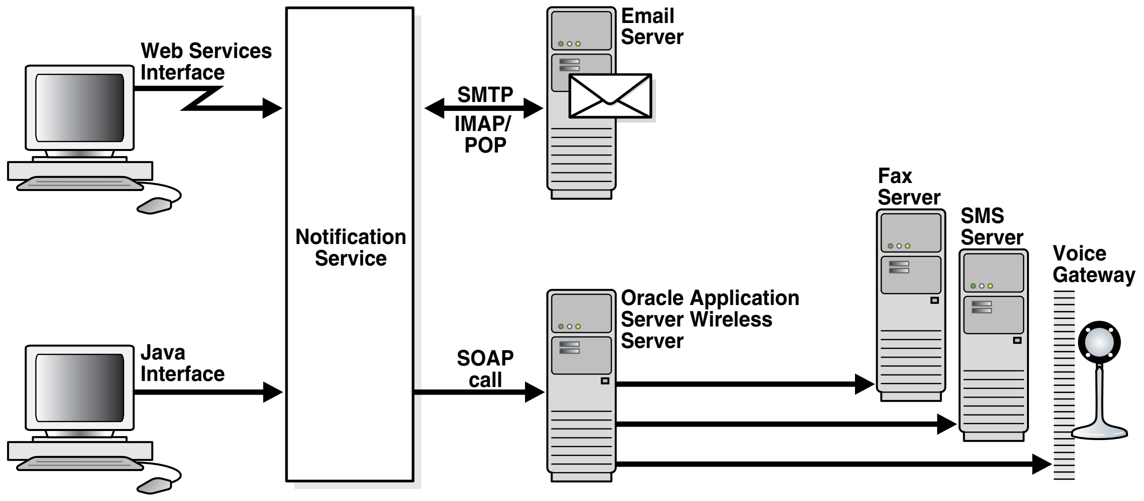
- Actionable notification—a notification to which the user can respond. For example, workflow sends an e-mail message to a manager to approve or reject a purchase order. The manager approves or rejects the request by replying to the e-mail with appropriate content.
- Oracle Application Server Wireless—the 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 or SMS.
  - 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 SMTP protocol and receiving e-mail from IMAP- and POP-based e-mail accounts.

Figure 15–1 shows the notification service interfaces and supported service types.

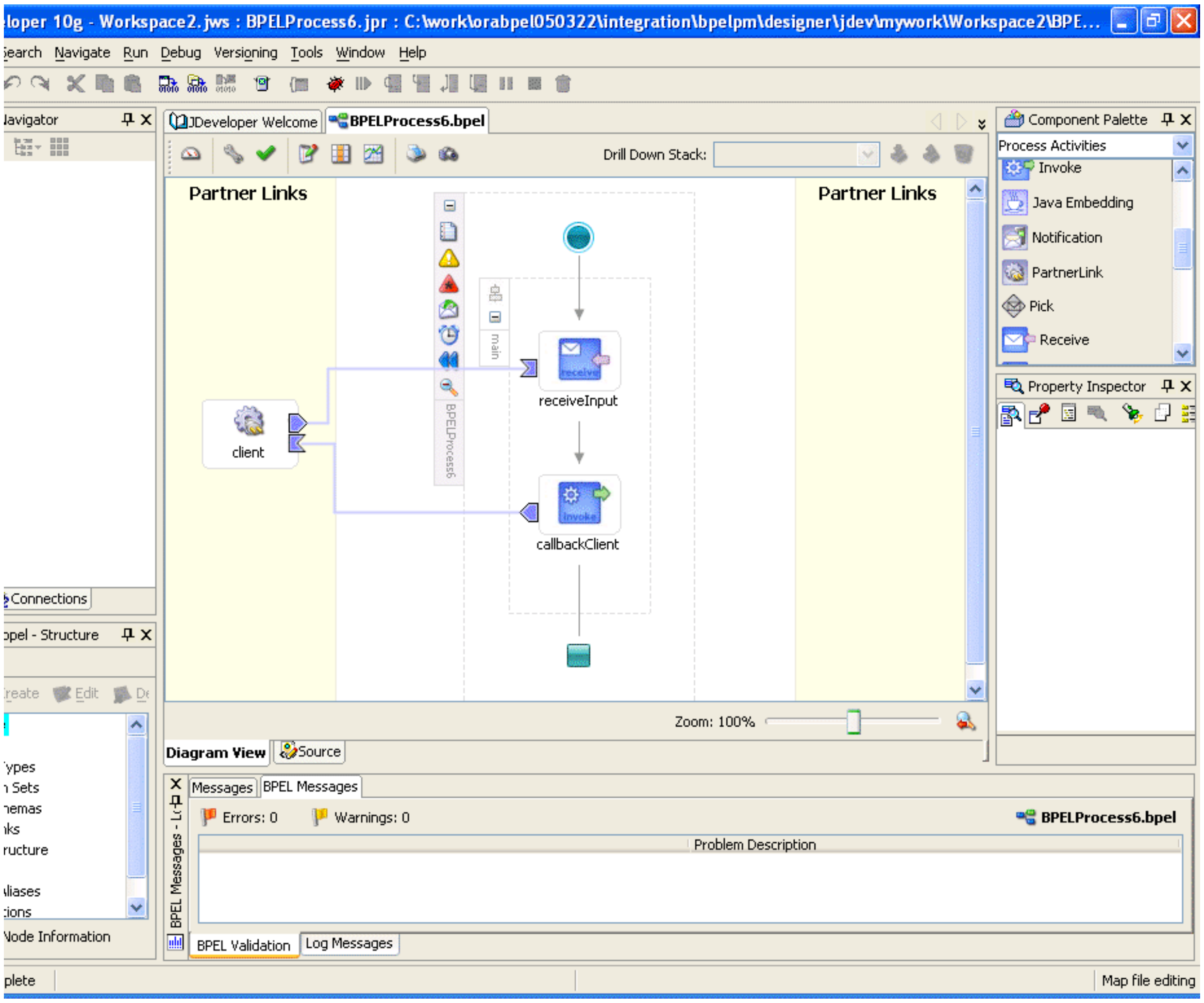
Figure 15–1 Notification Service Interfaces and Supported Service Types



Configuring Notification Service in JDeveloper BPEL Designer

The diagram view in JDeveloper BPEL Designer includes a **Notification** activity in the **Component Palette**, as shown in Figure 15–2.

Figure 15–2 Diagram View in JDeveloper BPEL Designer—Notification Activity

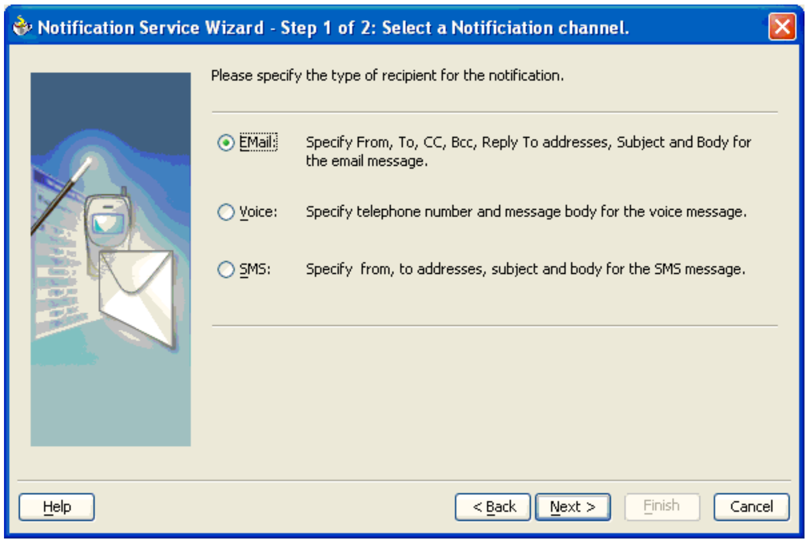


To use the notification service, do the following:

1. In **Diagram View**, select **Process Activities** from the **Component Palette** list.
2. Drag and drop a **Notification** activity from the **Process Activities** list to a position below **receiveInput** and above **callbackClient** in the diagram.

   The Notification Service wizard starts.
3. Click **Next** on the Welcome window.

   The Step 1 of 2: Select a Notification channel window appears.



Notification Service Wizard - Step 1 of 2: Select a Notification channel.

Please specify the type of recipient for the notification.

[✓ EMail:] Specify From, To, CC, Bcc, Reply To addresses, Subject and Body for the email message.

Voice: Specify telephone number and message body for the voice message.

SMS: Specify from, to addresses, subject and body for the SMS message.

4. Select a notification channel from the following options and click **Next**.
   - **EMail**
   - **Voice**
   - **SMS**

   The next step depends on which notification channel you select.

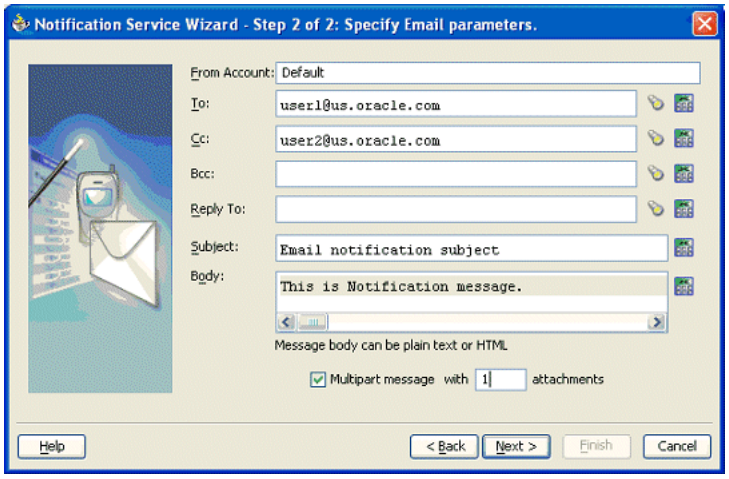
   See:

   - "The E-mail Notification Channel" on page 15-4 to configure e-mail notification
   - "The Voice Notification Channel" on page 15-10 to configure voice message notification
   - "The SMS Notification Channel" on page 15-12 to configure SMS notification

The E-mail Notification Channel

When you select **Email** for the notification channel, the Notification Service Wizard - Step 2 of 2: Specify Email Parameters window appears. Figure 15–3 shows the required e-mail notification parameters.

Figure 15–3 Notification Service Wizard - Step 2 of 2: Specify Email Parameters Window


Notification Service Wizard - Step 2 of 2: Specify Email parameters.

From Account: Default
To: user1@us.oracle.com
Cc: user2@us.oracle.com
Bcc:
Reply To:
Subject: Email notification subject
Body: This is Notification message.
Message body can be plain text or HTML
☑ Multipart message with 1 attachments

1. Enter information for each field as described in Table 15–1.

Table 15–1 E-mail Notification Parameters

Name	Description
From Account	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.
To	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 15-13.
CC and **Bcc**	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.
Reply To	The e-mail address to use for replies. This can be a static or dynamic address as described for the `To` address.
Subject	Subject of the e-mail message. This can be free text or dynamic text, or a combination. The XPath Expression Builder can be used to set dynamic text based on data from process variables that you specify. Dynamic data is automatically enclosed in `<% %>` when you use the XPath Expression Builder with free text.
Body	Message body of the e-mail message. This can be plain text, XML, free text, or dynamic text, as described for the **Subject** parameter.
Multipart message with n attachments	Select to specify e-mail attachments. See "Setting E-mail Attachments" on page 15-6. 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.

2. Click **Finish**.

   The BPEL fragment that invokes the notification service to send the e-mail message is created.

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. Run the Notification Service wizard and select **Email** for the channel.
2. Specify values for **To**, **Subject**, and **Body**.
3. Select **Multipart** and enter `2` for the number of attachments. (Note that the number of attachments must include the body part.)
4. The Notification Service wizard generates the `MultiPart` element with two body parts. The first body part is for the message body and the other is used for the attachment. The Notification Service wizard generates the BPEL fragment with an `assign` activity with multiple `copy` rules. One of the `copy` rules copies the attachment, as follows:

```
<assign name="Assign">
  <copy>
    <from expression="string('Default')"/>
    <to variable="varNotificationReq" part="EmailPayload"
query="/EmailPayload/ns1:FromAccountName"/>
  </copy>
...
<!--   copy statements relate to body and attachment -->
  <copy>
    <from>
      <Content xmlns="http://xmlns.oracle.com/ias/pcbpel/NotificationService">
        <MimeType
xmlns="http://xmlns.oracle.com/ias/pcbpel/NotificationService">multipart/mixed
        </MimeType>
        <ContentBody
xmlns="http://xmlns.oracle.com/ias/pcbpel/NotificationService">
          <MultiPart
xmlns="http://xmlns.oracle.com/ias/pcbpel/NotificationService">
            <BodyPart
xmlns="http://xmlns.oracle.com/ias/pcbpel/NotificationService">
              <MimeType
xmlns="http://xmlns.oracle.com/ias/pcbpel/NotificationService"/>
              <ContentBody
xmlns="http://xmlns.oracle.com/ias/pcbpel/NotificationService"/>
              <BodyPartName
xmlns="http://xmlns.oracle.com/ias/pcbpel/NotificationService"/>
            </BodyPart>
            <BodyPart
xmlns="http://xmlns.oracle.com/ias/pcbpel/NotificationService">
              <MimeType
xmlns="http://xmlns.oracle.com/ias/pcbpel/NotificationService"/>
              <ContentBody
xmlns="http://xmlns.oracle.com/ias/pcbpel/NotificationService"/>
              <BodyPartName
xmlns="http://xmlns.oracle.com/ias/pcbpel/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>
    <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>
    <from expression="string('NotificationAttachment2.html')"/>
    <to variable="varNotificationReq" part="EmailPayload"
query="/EmailPayload/ns1:Content/ns1:ContentBody/ns1:MultiPart/ns1:BodyPart[2]/
ns1:BodyPartName"/>
  </copy>
  <copy>
    <from expression="string('message2')"/>
    <to variable="varNotificationReq" part="EmailPayload"
query="/EmailPayload/ns1:Content/ns1:ContentBody/ns1:MultiPart/ns1:BodyPart[2]/
ns1:ContentBody"/>
  </copy>
</assign>
```

5. 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 `from expression` to copy `'image/gif'` as the `MimeType` (instead of the autogenerated `'text/html'`).

   b. Search for `BodyPart[2] BodyPartName` and change `from expression` to copy `'MyImage.gif'` (instead of the autogenerated `'NotificationAttachment2.html'`).

   c. Search for `BodyPart[2] ContentBody` and change `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 BPEL 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
:MimeType"/>
  </copy>
  <copy>
    <from expression="string('MyImage.gif')"/>
    <to variable="varNotificationReq" part="EmailPayload"
query="/EmailPayload/ns1:Content/ns1:ContentBody/ns1:MultiPart/ns1:BodyPart[2]/ns1
:BodyPartName"/>
  </copy>
  <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: `*Oracle_Home*\integration\orabpel\samples\tutorials\130.SendEmailWithAttachments` for an example of sending attachments using e-mail

Configuring the E-mail Server

The file `ns_emails.xml` in the directory `*Oracle_Home*\integration\orabpel\system\services\config` contains the configuration for e-mail accounts. Each `EmailAccount` element sets the configuration of a specific e-mail account. The `name` attribute in the `EmailAccount` element is the name of the account.

A default e-mail account is specified in the e-mail configuration file. This account is used when there is no account specified to send an e-mail notification. This account is also used to send task-related notifications. A default e-mail account must always be specified in the configuration file.

The `EmailAccount` element contains `OutgoingServerSettings` and `IncomingServerSettings` attributes. For actionable notifications in a workflow, both `IncomingServerSettings` and `OutgoingServerSettings` are required.

Table 15–2 describes the XML elements for the e-mail notification configuration stored in the `ns_emails.xml` file.

Table 15–2 XML Elements for the E-mail Notification Configuration File

Name	Description
`EmailAccount/Name`	Name of the account. This can be any name, but must be unique within this server.

Table 15–2 (Cont.) XML Elements for the E-mail Notification Configuration File

Name	Description
`EmailAccount/GeneralSettings/FromName`	Name of the `From` e-mail address
`EmailAccount/GeneralSettings/FromAddress`	E-mail address for the `From` e-mail address
`EmailAccount/OutgoingServerSettings/SMTPHost`	Name of the outgoing SMTP server
`EmailAccount/OutgoingServerSettings/SMTPPort`	Port of the outgoing SMTP server
`EmailAccount/IncomingServerSettings/Server`	Name of the incoming e-mail server
`EmailAccount/IncomingServerSettings/Port`	Port of the incoming e-mail server
`EmailAccount/IncomingServerSettings/UserName`	User ID of the e-mail address
`EmailAccount/IncomingServerSettings/Password`	User password
`EmailAccount/IncomingServerSettings/Password[encrypted`	Encrypted attribute of the password. It is `true` if the password is encrypted and `false` if it is not. Generally, you should set this to `false` when you first enter the password. The server automatically encrypts the password the first time it reads the configuration file and sets the attribute to `true`.
`EmailAccount/IncomingServerSettings/UseSSL`	Secure sockets layer (SSL) attribute. It is `true` if the incoming server requires SSL and `false` if it does not.
`EmailAccount/IncomingServerSettings/Folder`	Name of the folder from which to read the incoming messages
`EmailAccount/IncomingServerSettings/PollingFrequency`	Polling interval for reading messages from the incoming messages folder

Example ns_emails.xml File

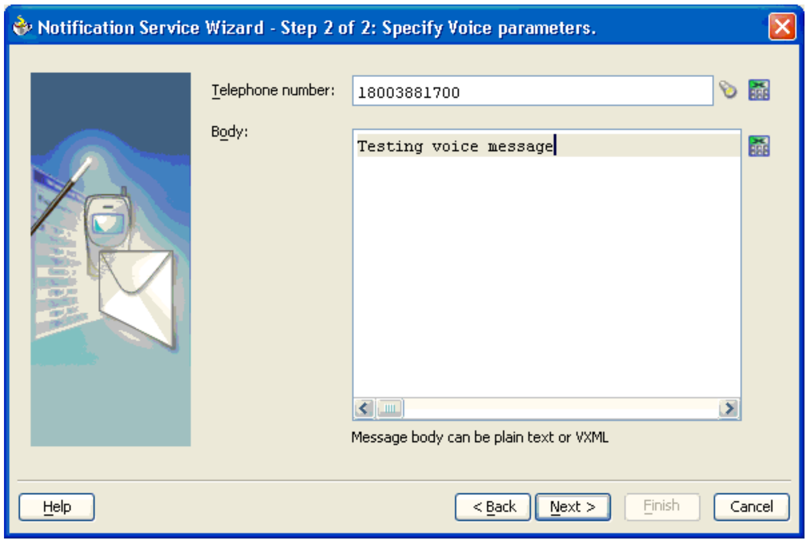
```
EmailAccounts xmlns="http://xmlns.oracle.com/ias/pcbpel/NotificationService">
   <EmailAccount>
       <Name>Default</Name>
      <GeneralSettings>
         <FromName>Oracle BPM</FromName>
         <FromAddress>bpm1@dlsun4254.us.oracle.com</FromAddress>
      </GeneralSettings>
      <OutgoingServerSettings>
         <SMTPHost>dlsun4254.us.oracle.com</SMTPHost>
         <SMTPPort>225</SMTPPort>
      </OutgoingServerSettings>
      <IncomingServerSettings>
         <Server>dlsun4254.us.oracle.com</Server>
         <Port>2110</Port>
         <Protocol>pop3</Protocol>
         <UserName>bpm1</UserName>
         <Password ns0:encrypted="false"
xmlns:ns0="http://xmlns.oracle.com/ias/pcbpel/NotificationService">welcome</Passwo
rd>
         <UseSSL>false</UseSSL>
```

```
                <Folder>Inbox</Folder>
                <PollingFrequency>1</PollingFrequency>
                <PostReadOperation>
                   <MarkAsRead/>
                </PostReadOperation>
             </IncomingServerSettings>
          </EmailAccount>
       </EmailAccounts>
```

The Voice Notification Channel

When you select **Voice** for the notification channel, the Notification Service Wizard - Step 2 of 2: Specify Voice Parameters window appears. Figure 15–4 shows the required voice notification parameters.

Figure 15–4 Notification Service Wizard - Step 2 of 2: Specify Voice Parameters Window


Notification Service Wizard - Step 2 of 2: Specify Voice parameters.

Telephone number: 18003881700

Body: Testing voice message

Message body can be plain text or VXML

1. Enter information for each field as described in Table 15–3.

Table 15–3 Voice Notification Parameters

Name	Description
Telephone number	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.
Body	Message body. This can be plain text or XML. Also, this can be free text or dynamic text, or a combination. The XPath Expression Builder can be used to set dynamic text based on data from process variables that you specify. Dynamic data is automatically enclosed in `<% %>` when you use the XPath Expression Builder with free text.

2. Click **Finish**.

   The BPEL fragment that invokes the notification service for voice notification is created.

Configuring the Wireless Service Provider for Voice

The configuration for the wireless service provider is stored in an XML file, `ns_iaswconfig.xml`, which is in

Oracle_Home`\integration\orabpel\system\services\config`

Table 15–4 describes the XML elements for the voice notification configuration stored in `ns_iaswconfig.xml` on the *Oracle_Home* server.

Table 15–4 XML Elements for the Voice Notification Configuration File

Name	Description
`/IASWConfiguration/SoapURL`	URL of the wireless service provider
`/IASWConfiguration/UserName`	Name of the user account with the wireless service provider
`/IASWConfiguration/Password`	User password
`/IASWConfiguration/Password[encrypted`	Encrypted attribute of the password. It is `true` if the password is encrypted and `false` if it is not. Generally, you should set this to `false` when you first enter the password. The server automatically encrypts the password the first time it reads the configuration file and sets the attribute to `true`.
`/IASWConfiguration/ProxyHost`	Name of the proxy server
`/IASWConfiguration/ProxyPort`	Port number of the proxy server

Note: The username and password are intentionally left blank at installation. If a username or password is not specified, the wireless server allows up to 50 notifications from a specific IP address. After 50 notifications, you must get a paid account from

`http://messenger.oracle.com`

Then you specify the appropriate username and password in the configuration file, `ns_iaswconfig.xml`, or by using Oracle Enterprise Manager 10*g* Application Server Control Console.

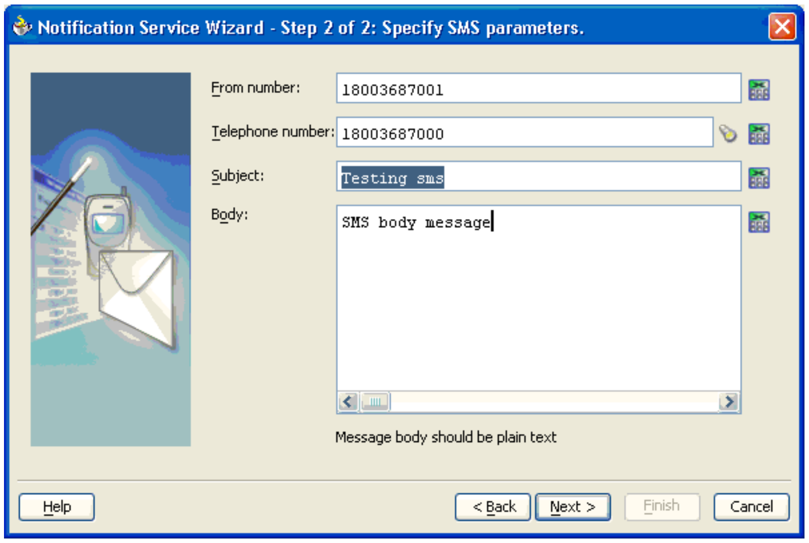
Example ns_iaswconfig.xml File

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<!--This XML file stores the details of the IAS Wireless Notification Service-->
<IASWConfiguration xmlns="http://xmlns.oracle.com/ias/pcbpel/NotificationService">
  <!-- URL to the SOAP Service -->
  <SoapURL>http://messenger.oracle.com/xms/webservices</SoapURL>

  <!-- UserName - this username should exist in iAS Wireless schema -->
  <UserName>username</UserName>

  <Password ns0:encrypted="false"
xmlns:ns0="http://xmlns.oracle.com/ias/pcbpel/NotificationService">password</Passw
ord>
</IASWConfiguration>
```

The SMS Notification Channel

When you select **SMS** for the notification channel, the Notification Service Wizard - Step 2 of 2: Specify SMS Parameters window appears. Figure 15–5 shows the required voice notification parameters.

Figure 15–5 Notification Service Wizard - Step 2 of 2: Specify SMS Parameters Window


Notification Service Wizard - Step 2 of 2: Specify SMS parameters.
Help
< Back
Next >
Finish
Cancel

From number: 18003687001
Telephone number: 18003687000
Subject: Testing sms
Body: SMS body message

Message body should be plain text

1. Enter information for each field as described in Table 15–5.

Table 15–5 SMS Notification Parameters

Name	Description
From number	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 15-13.
Telephone number	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.
Subject	Subject of the SMS message. This can be free text or dynamic text, or a combination. The XPath Expression Builder can be used to set dynamic text based on data from process variables that you specify. Dynamic data is automatically enclosed in `<% %>` when you use the XPath Expression Builder with free text.
Body	SMS message body. This must be plain text. This can be free text or dynamic text as described for the **Subject** parameter.

2. Click **Finish**.

   The BPEL fragment that invokes the notification service for SMS notification is created.

Configuring the Wireless Service Provider for SMS

As with the voice notification channel, the configuration for the wireless service provider for SMS is stored in the XML file `ns_iaswconfig.xml`, which is in

```
Oracle_Home\integration\orabpel\system\services\config
```

See "Configuring the Wireless Service Provider for Voice" on page 15-11 to configure a wireless service provider for SMS.

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:

  ```
  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`:

  ```
  <%bpws:getVariableData('inputVariable','payload','/client:BPELProcessRequest/cl
  ient: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:

  ```
  ora:getUserProperty(userID, propertyName)
  ```

  The first argument evaluates to the user ID. The second argument is the property name. Table 15–6 lists the property names that can be used in this XPath function.

Table 15–6 Properties for the Dynamic User XPath Function

Property Name	Description
`mail`	Look up a user’s e-mail address
`telephoneNumber`	Look up a user’s telephone number
`mobile`	Look up a user’s mobile telephone number
`homephone`	Look up a user’s home telephone number

The following example gets the e-mail address of the user identified by the variable `inputVariable`, part `payload`, and query `/client:BPELProcessRequest/client:userID`:

```
ora:getUserProperty(bpws:getVariableData(‘inputVariable’,
‘payload’,‘/client:BPELProcessRequest/client:userid’), ‘mail’)
```

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 "Selecting Users or Groups by Browsing the User Directory" on page 16-23 for more information on the Identity lookup dialog.

Starting Business Processes with the E-mail Activation Agent

You use the e-mail activation agent element `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 `activationAgents` element to `bpel.xml`.
   - See Table 15–7, " E-mail Activation Element and Respective Attributes in bpel.xml" and "The activationAgents Element Structure in bpel.xml" on page 15-14.
3. Include a corresponding account name configuration file in the project.
   - Name the file the same as the name of the `accountName` attribute of `activationAgents` in `bpel.xml`. See "The accountName XML File Structure" on page 15-14.

Table 15–7 describes the `activationAgents` element and `activationAgent` attributes of the activation fragment contained in the `bpel.xml` file.

Table 15–7 E-mail Activation Element and Respective Attributes in bpel.xml

Element/Attribute Name	Description
`/activationAgents/activationAgent[className]`	Name of the activation agent class. Use `com.collaxa.cube.activation.mail.MailActivationAgent` class as the activation agent.
`/activationAgents/activationAgent[heartBeatInterval]`	Polling interval of the messages in seconds
`/activationAgents/activationAgent/property name="accountName"`	Name of the e-mail configuration file. For example, if the account name is `test_account`, then the `test_account.xml` file is included in all the e-mail-related information.

The activationAgents Element Structure in bpel.xml

The following code example shows the structure of the `activationAgents` element contained in `bpel.xml`.

```
<activationAgents>
  <activationAgent

className="com.collaxa.cube.activation.mail.MailActivationAgent"
      heartBeatInterval="60">
     <property name="accountName">test_account</property>
  </activationAgent>
</activationAgents>
```

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>
```

Summary

This chapter describes how you can send an e-mail, voice, or short message service (SMS) message from Oracle BPEL Process Manager.

16 Oracle BPEL Process Manager Workflow Services

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:

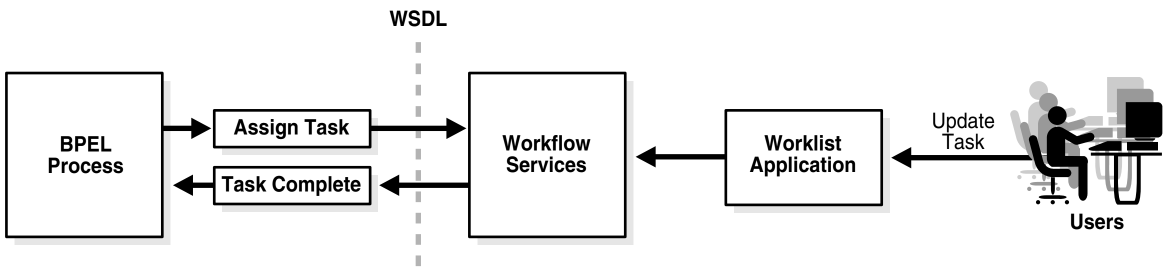
- Overview of Workflow Services
- Use Cases for Workflow Services
- Workflow Patterns
- Task Notifications
- Payload Display
- Configuration for Task Service
- Identity Service
- Workflow-Related XPath Extension Functions
- Approver Functions
- Vacation Request Example
- Summary

See Also: Appendix F, "Demo User Community" for the organizational hierarchy of the demonstration user community used in examples throughout this chapter

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 16–1, 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 (Worklist Application).

Figure 16–1 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
- Notification—an e-mail, voice, 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

Features of workflow services include:

- 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 and the ability to archive and purge task details based on specified policies
  - JSP-based forms for viewing and updating task details

  See Also: "Workflow Patterns" on page 16-8

- Multiple workflow patterns—includes standard patterns such as simple approval, sequential approval, parallel approval, and so on. Variations on workflow patterns such as automatic escalation, renewal, and reminders are also supported. You can also mix and match patterns to create complex patterns.

  See Also: "Workflow Patterns" on page 16-8

- Identity service—interacts with back-end identity management systems to capture all user information from Java AuthoriZatioN (JAZN) and LDAP. Identity services provide 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 16-75

- Notification service—notifies users of assigned tasks and task changes using various delivery channels, such as e-mail, voice message, or SMS. With notification service, you can also do the following:
  - Customize the notification content for different types of tasks
  - Perform actions on tasks using e-mail

  See Also: Chapter 15, "Oracle BPEL Process Manager Notification Service"

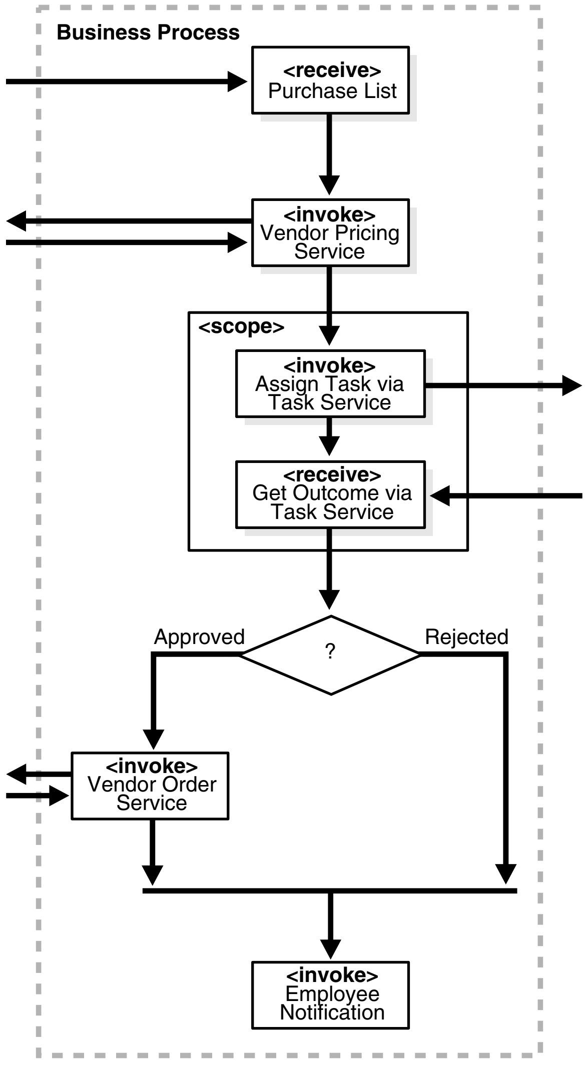
- The Worklist Application—enables you to access tasks and act on them. The Worklist Application can be extended or customized based on the application. With the Worklist Application, you can do the following:
  - View tasks assigned to a user or role
  - Perform authorized actions on tasks in the worklist
  - Filter tasks based on various criteria
  - Acquire and check out shared tasks

  See Also: Chapter 17, "Worklist Application"

Workflow Functionality: A Procurement Process Example

The functionality of workflow services can be illustrated using a simple procurement business process, as shown in Figure 16–2. The business process receives a purchase list of items from an employee and invokes the approved vendor's pricing service to determine the total cost to purchase those items. Then a task is assigned to the employee's manager. The manager reviews the total purchase cost and approves or rejects the request. The activities related to task assignment and getting the outcome of the task are performed within a scope to separate human workflow from the automated workflow. If the request is approved, then the vendor's order service is invoked to place the order for the requested items. The employee is notified whether the request is approved (with the order confirmation number received from the order service) or rejected. This requires a directory service lookup to determine user information. When the task is assigned to the manager, the manager may need to be notified that a task is waiting for approval. This requires the invocation of a notification service. The manager uses the Worklist Application to determine the details of the task (what approval is being requested and the amount requested) and then approves it.

Figure 16–2 BPEL Workflow


Business Process
<receive> Purchase List
<invoke> Vendor Pricing Service
<scope>
<invoke> Assign Task via Task Service
<receive> Get Outcome via Task Service
Approved
?
Rejected
<invoke> Vendor Order Service
<invoke> Employee Notification

Workflow Services Components

Figure 16–3 shows the following workflow services components:

- TaskActionHandler

  All workflow business processes use the services of a business process called TaskActionHandler to put items on a user's worklist and get responses from users. This process also maintains the timer events like task expiration, task reminder, and so on. This business process is predeployed in Oracle BPEL Server. See Appendix B, "Workflow and Notification Reference" for more information.

- TaskManagementService

  This 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/reassign tasks, and so on. The task service is used by the 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. Task service consists of the following services.

  See Appendix B, "Workflow and Notification Reference" for more information.

- TaskRoutingService

  The TaskRoutingService offers services to route, escalate, and reassign the task. See Appendix B, "Workflow and Notification Reference" for more information.

- Identity service

  Identity service is a thin Web service layer on top of the Oracle Application Server 10*g* 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. See "Identity Service" on page 16-75 for more information.

- Worklist Application

  The Worklist Application is a sample Web-based application that provides the ability to retrieve tasks based on a variety of filter criteria and to perform task actions on the selected tasks. See Chapter 17, "Worklist Application" for more information.

- Notification service

  Notification service delivers notifications with the specified content to the specified user to any of the following channels: e-mail, telephone voice message, and SMS. See Chapter 15, "Oracle BPEL Process Manager Notification Service" for more information.

Figure 16–3 Workflow Services Components

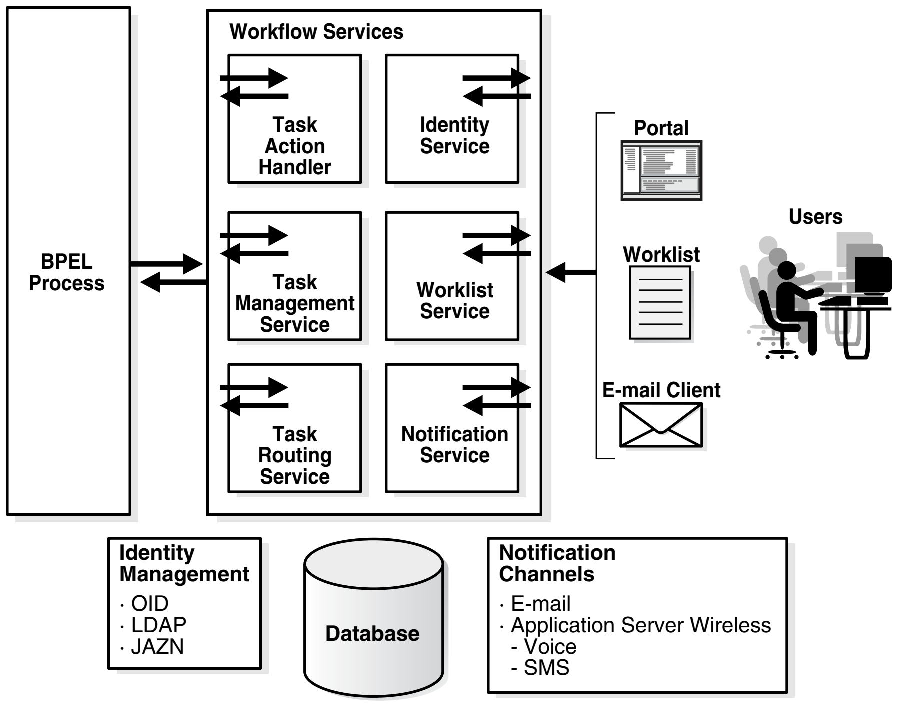
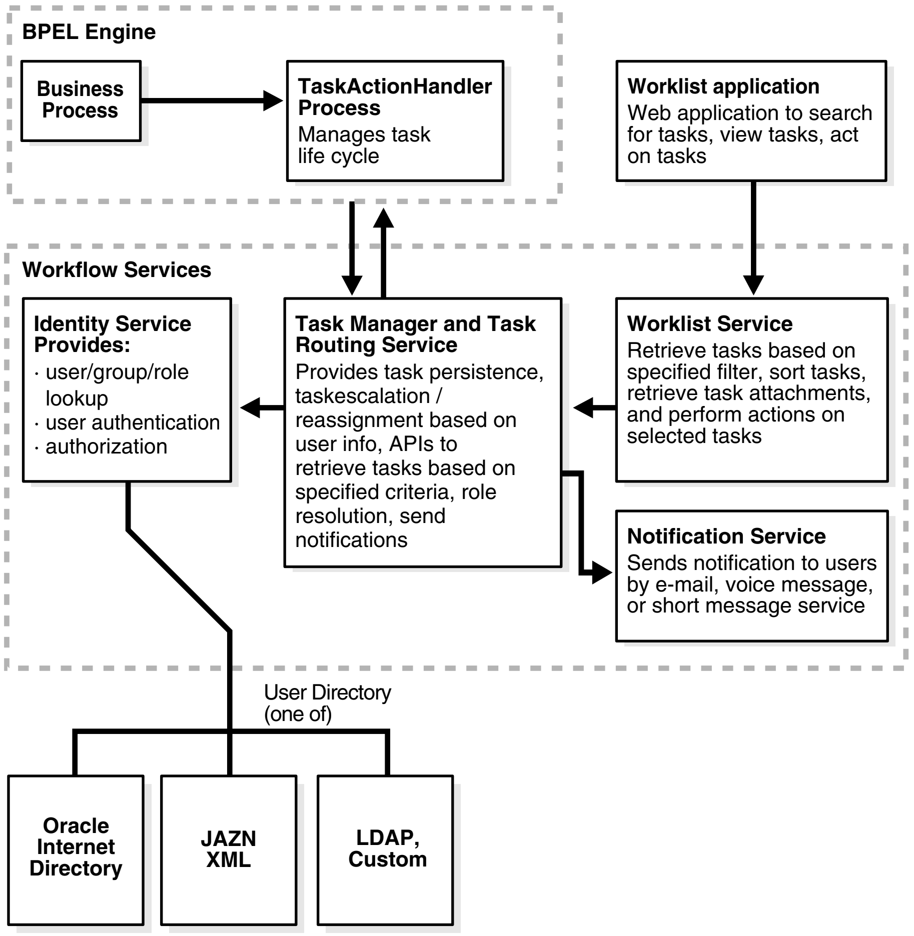


Figure 16–4 shows the interactions between the services and the business process.

Figure 16–4 Workflow Services and Business Process Interactions


BPEL Engine
Business Process
TaskActionHandler Process
Manages task life cycle
Worklist application
Web application to search for tasks, view tasks, act on tasks
Workflow Services
Identity Service Provides:
· user/group/role lookup
· user authentication
· authorization
Task Manager and Task Routing Service
Provides task persistence, taskescalation / reassignment based on user info, APIs to retrieve tasks based on specified criteria, role resolution, send notifications
Worklist Service
Retrieve tasks based on specified filter, sort tasks, retrieve task attachments, and perform actions on selected tasks
Notification Service
Sends notification to users by e-mail, voice message, or short message service
User Directory (one of)
Oracle Internet Directory
JAZN XML
LDAP, Custom

Use Cases for Workflow Services

Using workflow services is demonstrated in the VacationRequest, LoanDemoPlusWithWorkflow, DocumentReview, and HelpDeskServiceRequest demos.

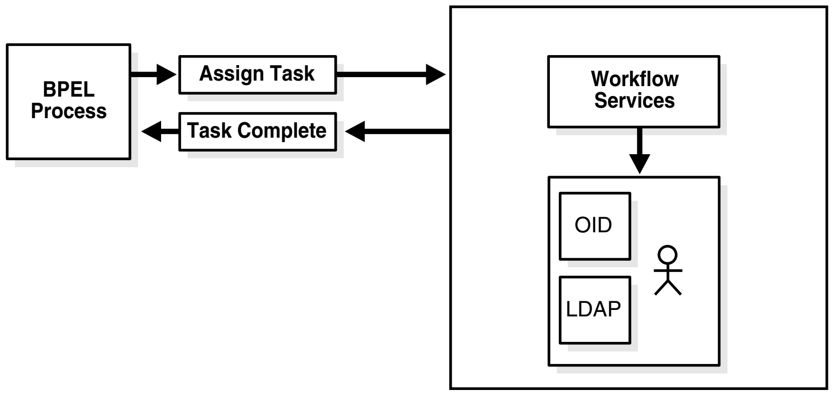
> **See:** `Oracle_Home\integration\orabpel\samples\demos`

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 16–5, is described in the VacationRequest sample.

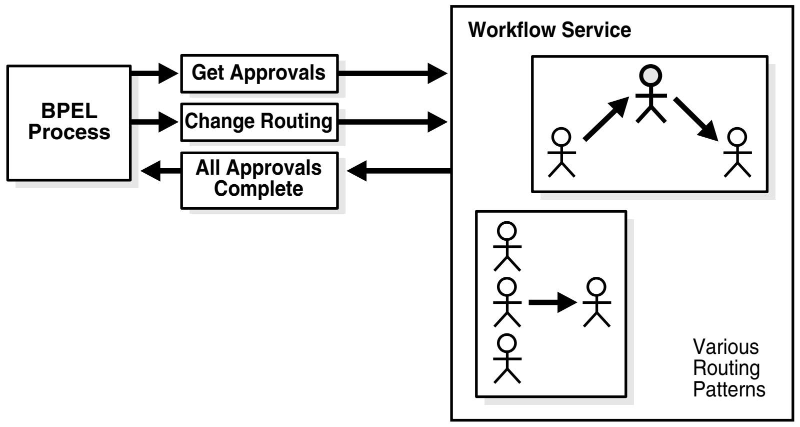
Figure 16–5 Assigning Tasks to a User or Role from a Directory



Using the Various Flow Patterns

A task may be routed through multiple users through a sequential flow, a parallel flow, or an adhoc flow. 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 that $100,000. This scenario, shown in Figure 16–6, is described in the LoanDemoPlusWithWorkflow sample.

Figure 16–6 Flow Patterns and Routing Policies

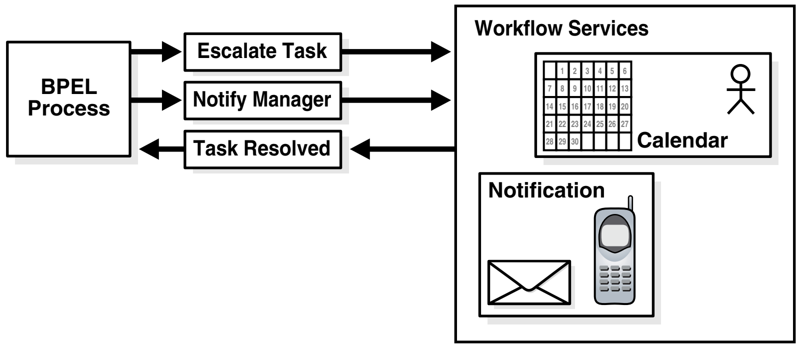


See "Workflow Patterns" on page 16-8 for the various flow patterns supported by workflow services. You can use these patterns as building blocks to create complex workflows.

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 or telephone voice message. 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 16–7. 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 2 days, then the task is routed to the manager for further action.

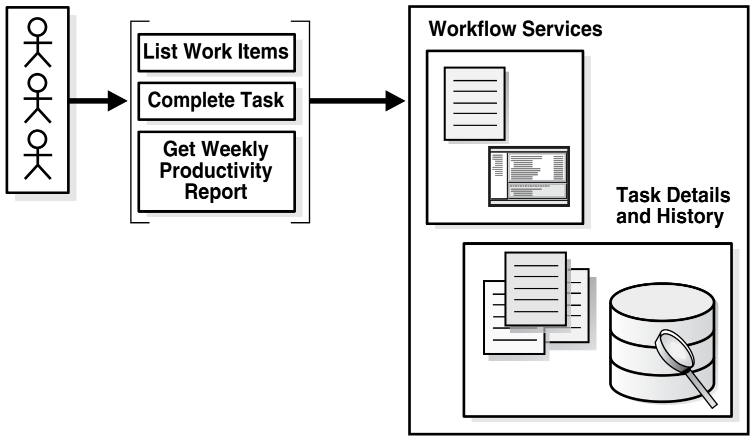
Figure 16–7 Escalation and Notification



The Worklist Application

Users typically access tasks assigned to them by using the Worklist Application, as shown in Figure 16–8. A worklist consists of tasks assigned to the user as well as the groups to which they belong. A task may also include forms and attachments in addition to other task details such as history, comments, and approval sequence. The worklist may also be accessed from Oracle Portal or other clients to act on tasks as well as get productivity reports. The Worklist Application can be customized and extended based on the specific needs of an application. See Chapter 17, "Worklist Application" for details about worklist functionality and the sample Worklist Application.

Figure 16–8 Oracle BPEL Worklist Application—Access Tasks, Forms, Attachments, and Reports



Workflow Patterns

Oracle BPEL Process Manager provides a library of workflow patterns. You can choose a pattern that meets your business requirement and model your workflow based on the pattern. Oracle BPEL Process Manager supports the following patterns:

- Simple workflow (single-user task)—used for a business process that requires a user's action (or inaction if the user does not act on the task within the allotted time). Based on the user's action or inaction, the business process modeler defines what the business process does. See "Simple Workflow" on page 16-31 for more information.

- Simple workflow with escalation (extension of a single-user task)—used to escalate the task, to the user's manager for example, if the user does not respond within the allotted time. See "Simple Workflow with Automatic Escalation" on page 16-33 for more information.
- Simple workflow with renewal (extension of a single-user task)—used to extend the expiration period if the user does not respond within the allotted time. The business process modeler specifies how many times the task can be renewed before it expires. See "Simple Workflow with Automatic Renewal" on page 16-37 for more information.
- Sequential workflow—used to route tasks to multiple users in a sequence. The business process modeler specifies the participants for the task as a list or a management chain. See "Sequential Workflow" on page 16-40 for more information.
- Sequential workflow with escalation (extension of a sequential workflow)—used to escalate the task if a user does not take action within the allotted time. See "Sequential Workflow with Escalation" on page 16-46 for more information.
- Parallel workflow—used when multiple users, working in parallel, must take action, such as in a hiring situation when multiple users vote to hire or reject an applicant. The business process modeler specifies the voting percentage that is needed for the outcome to take effect, for example, a majority vote or a unanimous vote. See "Parallel Workflow" on page 16-49 for more information.
- Parallel workflow with final reviewer (extension of parallel workflow)—used when a task is first sent to multiple users in parallel and then sent to a final reviewer for a decision. See "Parallel Workflow with Final Reviewer" on page 16-52 for more information.
- Adhoc (or dynamic) workflow—used to assign a task to one user first, who then decides where the task goes next. The task is routed until one of the assignees completes it and does not route it further. See "Adhoc Workflow" on page 16-54 for more information.
- FYI task—used when a task is sent to a user, but the business process does not wait for a user response; it just continues. See "FYI Tasks" on page 16-55 for more information.
- User Task 2.0 Macro—supports user tasks from Oracle BPEL Process Manager release 2.0. This user task requires the business process modeler to explicitly assign task properties and also requires a custom application to view and act on tasks. The User Task 2.0 Macro is available for backward compatibility and is replaced with the new workflow services and patterns in this release. See "The User Task 2.0 Macro" on page 16-56 for more information.
- Task continuation—used to build complex workflow patterns using the task continuation (extension) concept. Task continuation allows one workflow to be continued with another workflow, thus creating a mix of the previously described patterns to create complex workflows. See "Task Continuations" on page 16-56 for more information.

The Modeling Process

Using the JDeveloper BPEL Designer component of Oracle BPEL Process Manager, you can model workflow by specifying parameters and answering questions posed by the Workflow wizard.

Modeling a workflow is a five-step process:

1. Specify the workflow pattern.
2. Specify task details and configurations.

   Add the task title, payload, expiration parameter, outcomes, and so on.
3. Specify the assignment policy.

   Assign the task to the user, role, or group, and indicate whether it is a static or dynamic assignment.

   See "Task Assignment" on page 16-19 for more information.
4. Specify the pattern-specific policies.

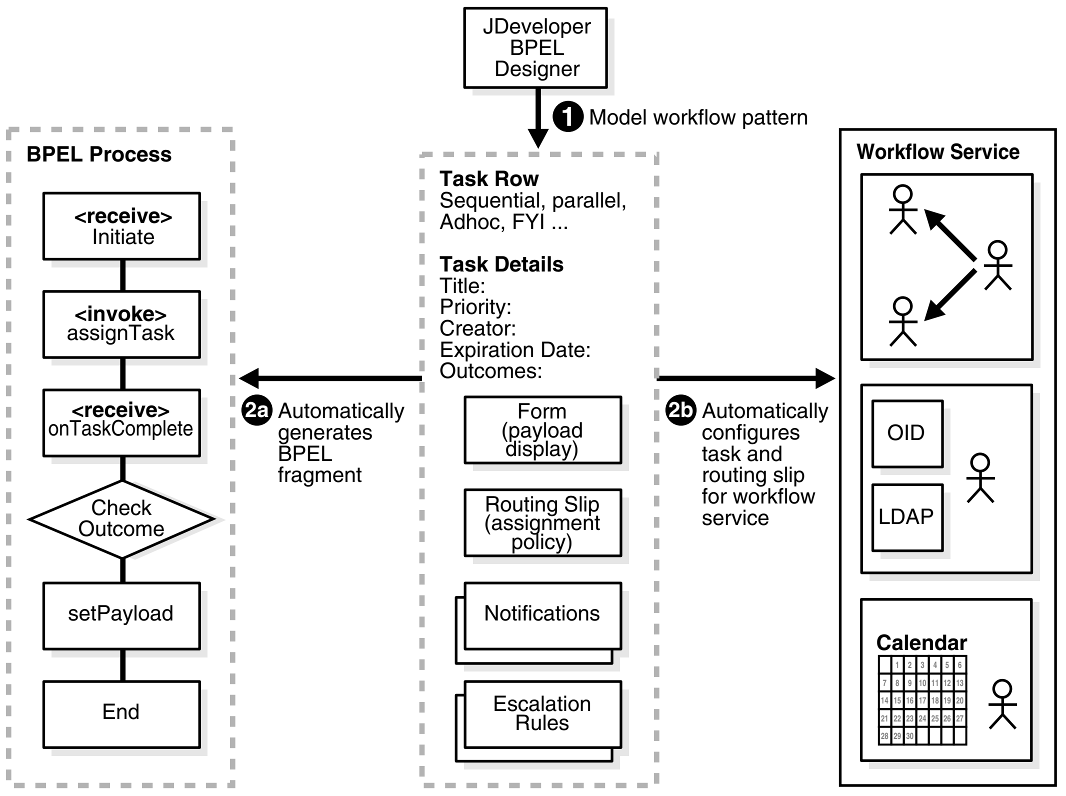
   Each pattern defines policies specific to the pattern. For example, sequential workflow requires routing rules, which may be based on management levels, titles, and the outcomes that cause a task to be routed further.
5. Specify task notification.

   Notifications are sent to alert users of changes to the state of a task. Notifications can be sent by e-mail, telephone voice message, or SMS.

   See "Task Notifications" on page 16-60 and Chapter 15, "Oracle BPEL Process Manager Notification Service" for more information.

The end result of specifying the parameters is the generation of a BPEL scope that uses BPEL constructs to orchestrate the workflow. The Workflow wizard in JDeveloper BPEL Designer automatically generates the BPEL fragment and task configuration, as shown in Figure 16–9.

Figure 16–9 Pattern-Based Modeling



Editing or Deleting a Workflow

To edit a workflow, you manually edit the invoke and assign activities that are created when you configure the workflow pattern. For each workflow pattern described in this chapter, see the discussion on customization for more information.

To delete a workflow, you delete the scope and search block, which automatically deletes the variables, partner links, and so on.

Task Details and Configurations

When you model a workflow, you specify the values for task attributes such as title, payload, expiration, and configuration for the task. The configuration includes possible outcomes of the task, payload display, and so on. The task attributes are assigned using the BPEL assign activity. The task configuration is captured in an XML file. The XML file is named `taskConfig`*`workflow_name`*`.xml` and is available as part of the JDeveloper project.

Task Attributes

The following task attributes are shown in Figure 16–10 and Figure 16–11.

- **Task Title** (required)

  The task title is used to display the task in Oracle BPEL Worklist Application. The title can include static strings and data from other business process variables.

- **Payload** (required)

  The task payload is the data in the task. The payload is restricted to BPEL variables and its substructures.

- **Payload Display** (required)

  The payload display identifies the display mechanism for the payload in the Worklist Application. **Auto generated JSP form** is the default option in which a JSP is generated by inferring the schema of the selected payload. Use the **XSL file** option to specify an XSL file for the payload display. Use the **JSP URL** option to specify your own JSP that can be used to display the payload. See "Payload Display" on page 16-64 for more information.

- **Task Creator** (optional)

  The creator of the task is the user who initiates the task. The creator can be defined using the XPath Expression Builder. The creator can view the tasks they created from the Worklist Application if they have access to the application. The creator can perform some actions specific to the creator from the Worklist Application, such as withdraw the task. If the creator is not specified, it defaults to the task owner.

- **Expiration Duration**

  The expiration duration is the maximum duration that the task can be open. This is optional for some workflow patterns and is required for other workflow patterns (such as auto-escalate).

- **Task Priority**

  If you select the advanced options in the Workflow wizard, you can 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.

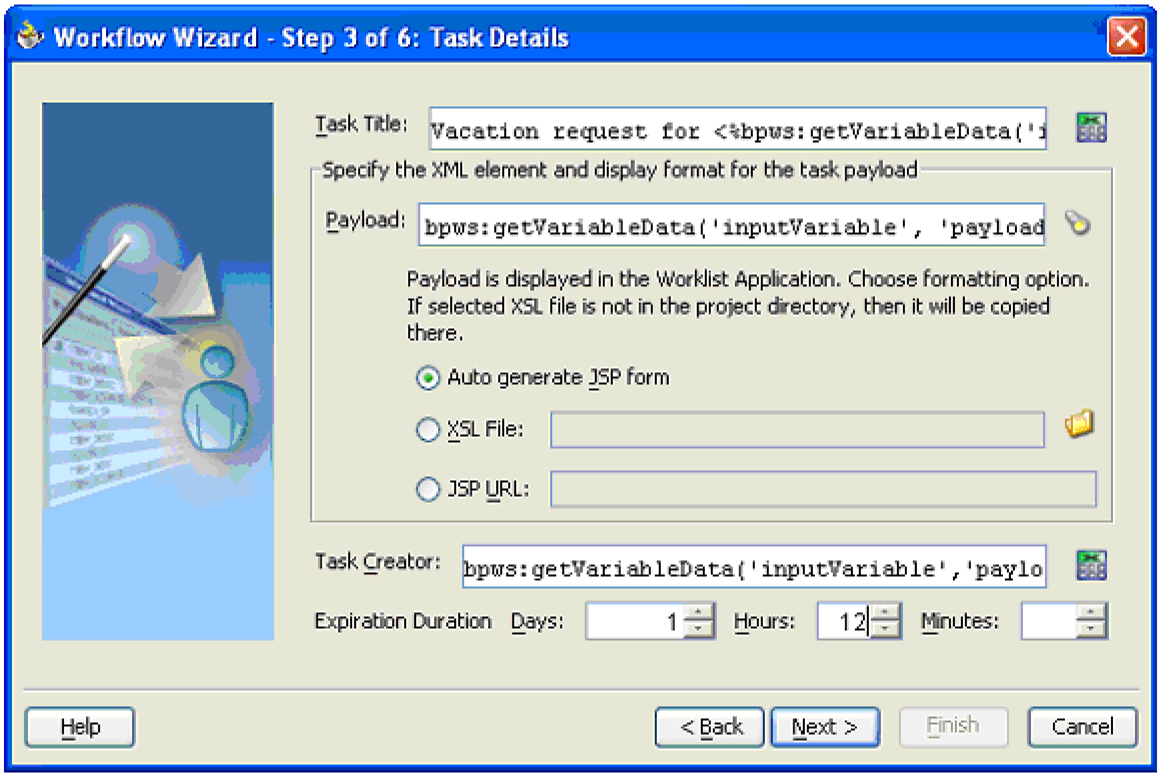
- **Task Owner** (optional)

  The task owner can view the 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. This optional attribute defaults to the system user `bpeladmin` if not specified. The task owner can be specified in the advanced screens.

- **Identification Key** (optional)

  The identification key can be used as a user-defined identification 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 Worklist Application by the identification key. This attribute has no default value.

Figure 16–10 Workflow Wizard: Task Details



Task Title: Vacation request for <%bpws:getVariableData('i

Specify the XML element and display format for the task payload

Payload: bpws:getVariableData('inputVariable', 'payload

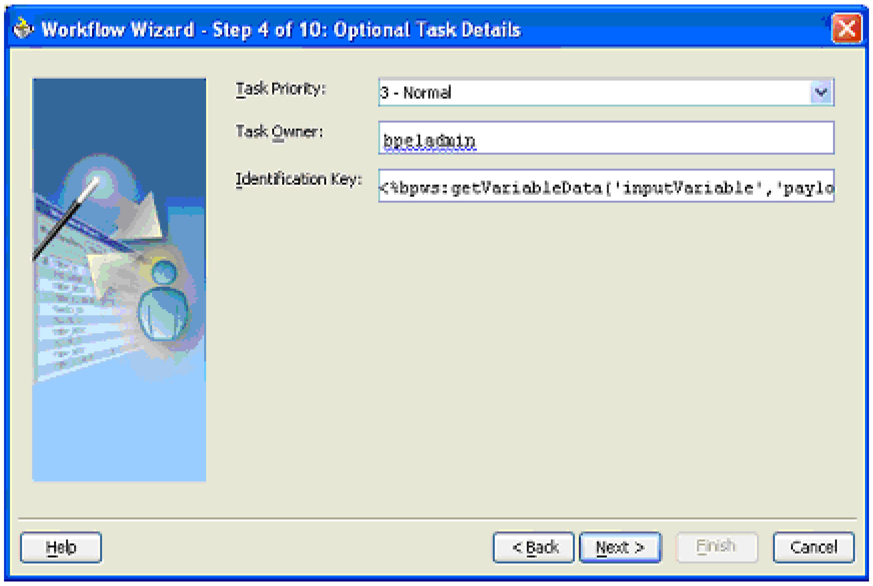
Payload is displayed in the Worklist Application. Choose formatting option. If selected XSL file is not in the project directory, then it will be copied there.

[✓ Auto generate JSP form] XSL File: JSP URL:

Task Creator: bpws:getVariableData('inputVariable','paylo

Expiration Duration Days: 1 Hours: 12 Minutes:

Figure 16–11 Workflow Wizard: Optional Task Details



Task Priority: 3 - Normal

Task Owner: bpeladmin

Identification Key: <%bpws:getVariableData('inputVariable','paylo

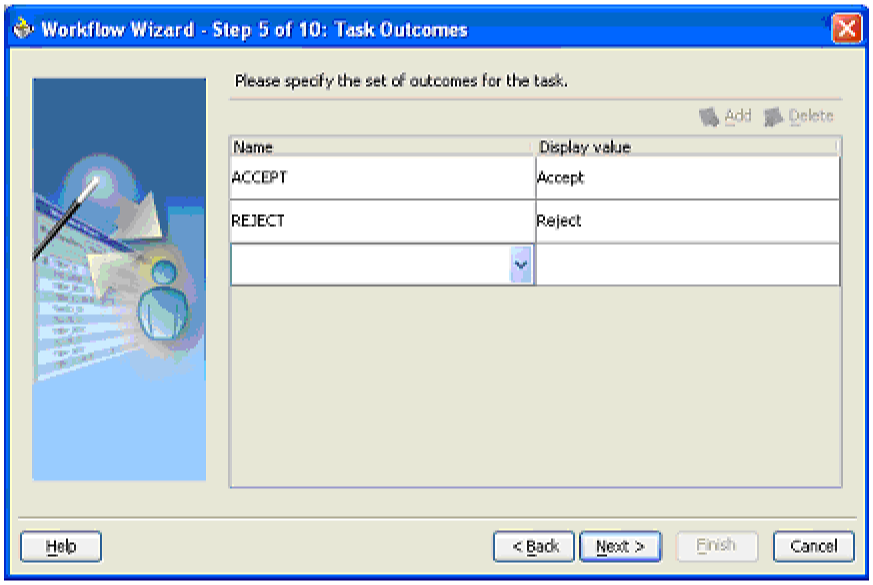
Task Outcomes

The task outcomes capture the possible outcomes of a task such as Accept, Reject, and Approve. The outcomes of the task are specified during the creation of the workflow. The Worklist Application displays these outcomes as the possible actions that a user can perform. You can select one of the seeded outcomes as the possible actions that a user can perform. The task outcomes can also have display values that are different from the actual outcome value. This permits outcomes to be internationalized. See "Resource Bundles" on page 16-19 for more information about internationalization.

Figure 16–12 shows where the conclusions are added. By default, JDeveloper BPEL Designer displays common outcomes from the following file, which can be changed to modify the default list of outcomes:

Oracle_Home\integration\jdev\jdev\system10.1.2.0.0.1811\TaskConclusion.xml

Figure 16–12 Task Outcomes



Please specify the set of outcomes for the task.

Name	Display value
ACCEPT	Accept
REJECT	Reject

After the workflow is created, you can change the custom actions in the `conclusions` element of the task configuration XML file, as shown in bold in the following code example.

```
<taskType ....>
  <task ...>
    <conclusions>
      <conclusion name="ACCEPT">
        <displayValue>Accept</displayValue>
      </conclusion>
      <conclusion name="REJECT">
        <displayValue>Reject</displayValue>
      </conclusion>
    </conclusions>
        ...
  </task>
  <notifications>
      ...
  </notifications>
</taskType>
```

Advanced Task Configurations

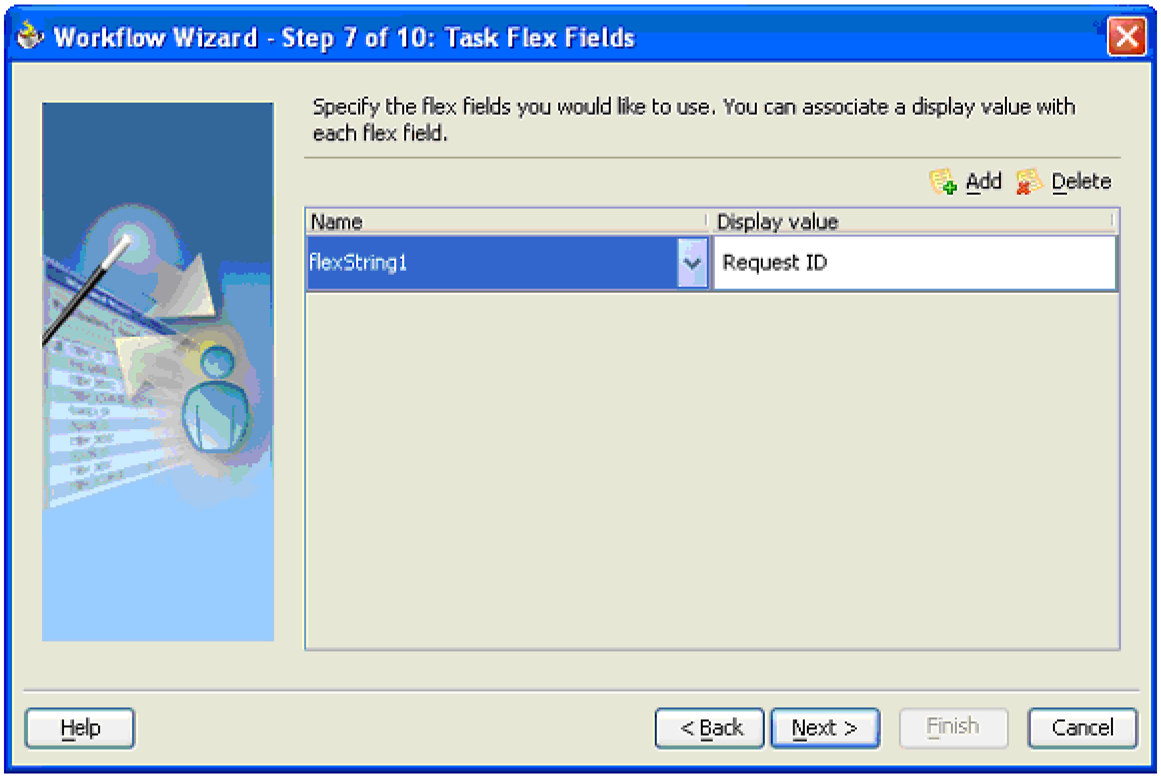
The task also has the following optional configurations. All these configurations are captured in the task configuration file. The following configurations are available through the advanced options in the workflow.

- Flex Fields
- Restricted Actions

- Version-Tracking Attributes
- Task Notifications and Reminders
- Resource Bundles

Flex Fields The task object contains flex fields for extending the task to capture any data in addition to the payload. These flex fields are treated like other header attributes in the worklist. Tasks can be searched based on data in any of the flex fields. Flex fields are available for each of the following data types: `string`, `double`, `long`, and `date`. When modeling the business process, you can specify which flex fields are used and a display string for the flex fields. The display value is used by the Worklist Application for the data (instead of using the name of the flex field itself). This also permits display values to be internationalized. See "Resource Bundles" on page 16-19 for more information on internationalization. Figure 16–13 shows how you configure flex fields. Only the flex fields that are configured in this window are displayed in the Worklist Application.

Figure 16–13 Task Flex Fields



Specify the flex fields you would like to use. You can associate a display value with each flex field.

Name	Display value
flexString1	Request ID

You can change flex field information in the `flexFields` element of the task configuration XML file, as shown in bold in the following code example.

```
<taskType ....>
  <task ...>
    ...
    <flexFields>
      <flexField name="flexString2">
        <displayValue>displayfor flex string</displayValue>
      </flexField>
    </flexFields>
   ...
  </task>
  <notifications>
      ...
  </notifications>
</taskType>
```

The following flex fields are available:

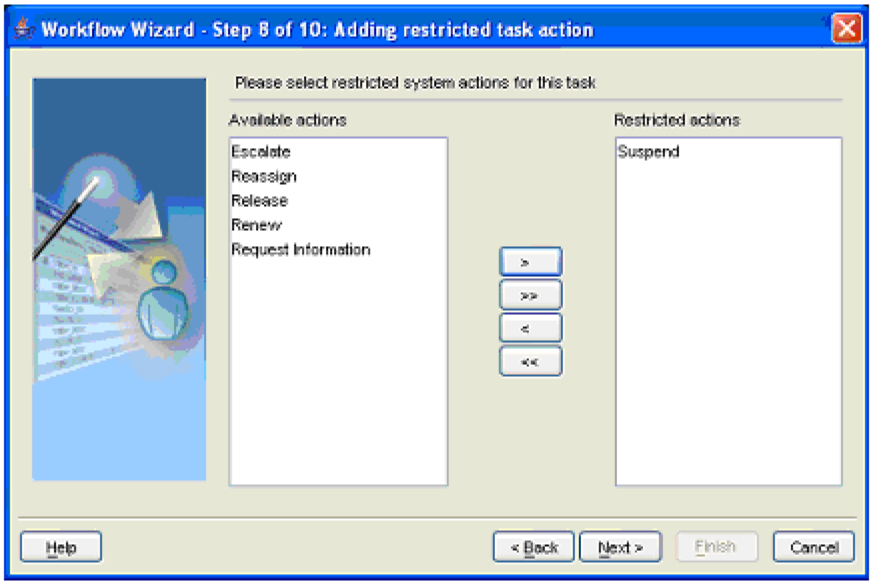
- `flexString1`
- `flexString2`
- `flexString3`
- `flexString4`
- `flexLong1`
- `flexLong2`
- `flexDouble1`
- `flexDouble2`
- `flexDate1`
- `flexDate2`
- `flexDate3`

The Workflow wizard in JDeveloper BPEL Designer does not capture the assignment to the flex fields. The assignments to the flex fields can be manually added in the workflow BPEL scope. In every workflow that is generated, an `assign` named `setUserDefinedAttributes` in the scope is generated for the workflow. The assignments to the flex fields can be added in this `assign`, as shown in the following BPEL code example.

```
<sequence>
  <assign name="setUserDefinedAttributes">
       ...
    <copy>
      <from expression="string('value of flex string 1')"/>
      <to variable="WorkflowVar1" query="/task:task/task:flexString1"/>
    </copy>
  </assign>
     ...
<sequence>
```

Restricted Actions The actions that are performed from the 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 never to suspend a loan application. To model this scenario, at design time, you can mark `Suspend` as a restricted action. When an action is marked as restricted, the Worklist Application does not display the action for this task. Figure 16–14 shows how restricted actions are configured.

Figure 16–14 Restricted Actions



The following actions can be restricted:

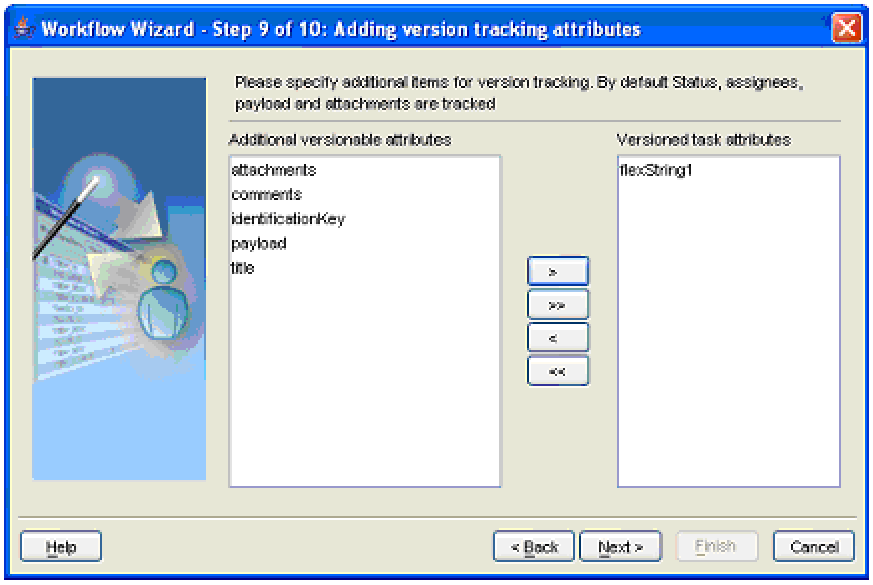
- Auto Release
- Reassigned
- Escalated
- Renewed
- Info Requested

After the workflow is created, you can configure restricted actions in the `restrictedActions` element of the task configuration XML file, as shown in the following code example.

```
<taskType ...>
  <task ...>
    ...
    <restrictedActions>
      <restrictedAction>AUTO RELEASE</restrictedAction>
      <restrictedAction>ESCALATED</restrictedAction>
      <restrictedAction>REASSIGNED</restrictedAction>
      <restrictedAction>RENEWED</restrictedAction>
      <restrictedAction>INFO REQUESTED</restrictedAction>
      <restrictedAction>SUSPENDED</restrictedAction>
    </restrictedActions>
  </task>
  <notifications>
     ...
  </notifications>
</taskType>
```

Version-Tracking Attributes When a task is modified, Oracle BPEL Process Manager creates a new version of the task as part of the task history. Changes to some of the task attributes, such as status, assignees, payload, and attachments, are always tracked with a new version. In addition, you can mark other attributes of the task as version tracked, including the title of the task, any flex attributes, and comments. Figure 16–15 shows where the version tracking attributes are specified.

Figure 16–15 Version Tracking Attributes


Workflow Wizard - Step 9 of 10: Adding version tracking attributes
Please specify additional items for version tracking. By default Status, assignees, payload and attachments are tracked
Additional versionable attributes
attachments
comments
identificationKey
payload
title
Versioned task attributes
flexString1
>
>>
<
<<
Help
< Back
Next >
Finish
Cancel

The following attributes can be version tracked:

- `flexString1`
- `flexString2`
- `flexString3`
- `flexString4`
- `flexLong1`
- `flexLong2`
- `flexDouble1`
- `flexDouble2`
- `flexDate1`
- `flexDate2`
- `flexDate3`
- `title`
- `payload`
- `identificationKey`
- `comments`
- `attachments`

After the workflow is created, you can change version tracked attributes in the `versionTracking` element of the task configuration XML file, as shown in the following code example.

```
<taskType ....>
  <task ...>
    ...
    <versionTracking>
      <attribute>attachments</attribute>
      <attribute>payload</attribute>
      <attribute>flexString2</attribute>
      <attribute>comments</attribute>
    </versionTracking>
      ...
  </task>
  <notifications>
   ...
  </notifications>
</taskType>
```

Task Notifications and Reminders As part of workflow modeling, you can optionally specify notifications and reminders to be sent to users if the task is not acted upon in a certain time. See "Task Notifications" on page 16-60 for more information.

Resource Bundles Resource bundles can be specified in the task configurations. Resource bundles are used for the following task configurations:

- Display value for outcomes

  Display values for the outcomes can be plain text or keys in the resource bundle specified. When the display value is for the format `message(key)`, the display value is fetched from the resource bundle specified for the task configuration.

- Display value for flex strings

  Display values for the flex strings are treated like the display value for task outcomes.

- Notification messages

  Notification messages for e-mails can also be internationalized. To get the internationalized message for notifications, use the XPath extension function `orcl:get-localized-string()`.

The resource bundle can be part of the BPEL suitcase or in some other location that can be looked up by the BPEL run-time server. To make the resource bundle available as part of the BPEL suitcase, add the resource bundle files to the project before deploying the project.

Resource bundle information is captured in the task configuration file as shown in the following code example. If the resource bundle is available as part of the project, then `resourceBundleLocation` need not be set.

```
<taskType ...
        resourceBundleName="VacationRequestResource"
        resourceBundleLocation=" VacationRequestResourceLocation"
        ...
 ...
</taskType>
```

Task Assignment

Tasks can be assigned to both groups and users, as shown in Figure 16–16. When tasks are assigned to groups, the task can be seen by any user in the group. For a user to act on a task assigned to a group, the user must acquire the task first. Task assignees can be specified in one of the following ways.

- Static assignment

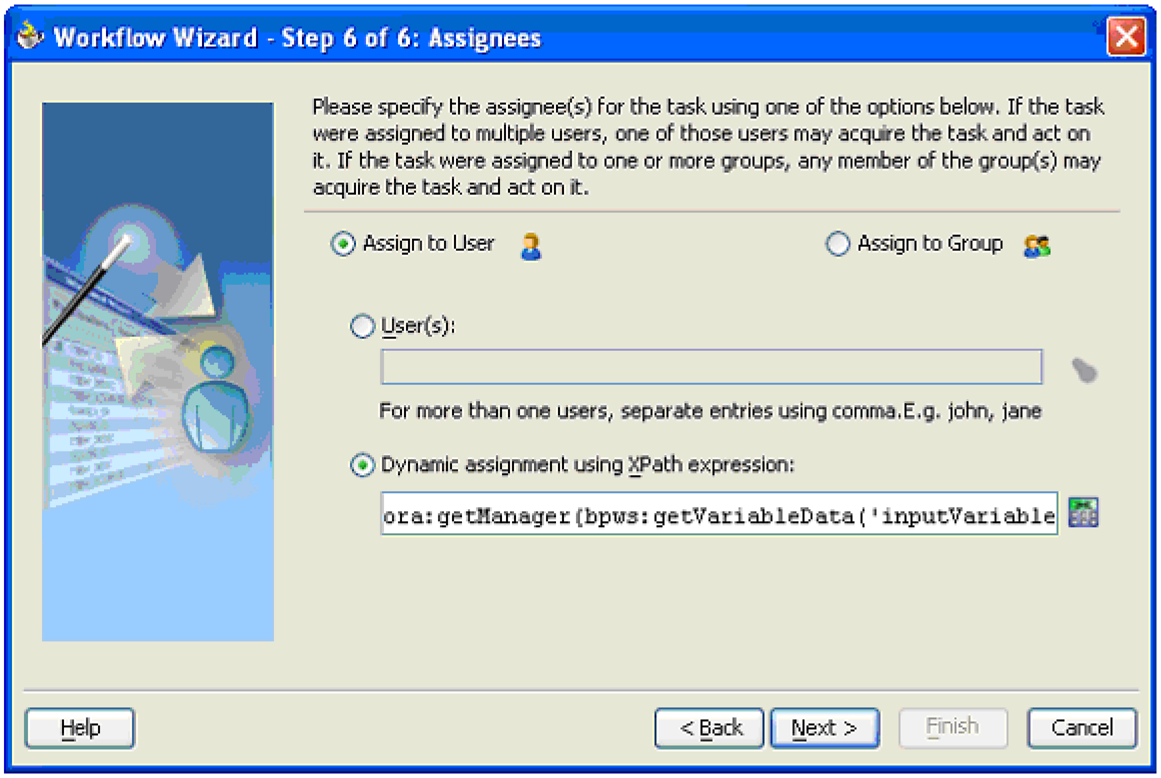
  In a static assignment, the assignees are specified at design time. The users can be typed in or they can be selected by browsing the identity service-supported user directories.

- Dynamic assignment using XPath expressions

  The assignees of the task can be assigned from other business process variables using the `bpws:getVariableData(...)` expression and other XPath expressions. For example, the task can be assigned to the manager of the vacation requester using the expression

  ```
  ora:getManager(bpws:getVariableData('inputVariable','payload','/client:
  VacationRequestProcessRequest/client:creator'))
  ```

Figure 16–16 Dynamic Assignment Using an XPath Expression



Please specify the assignee(s) for the task using one of the options below. If the task were assigned to multiple users, one of those users may acquire the task and act on it. If the task were assigned to one or more groups, any member of the group(s) may acquire the task and act on it.

[✓ Assign to User] Assign to Group

User(s):

For more than one users, separate entries using comma.E.g. john, jane

[✓ Dynamic assignment using XPath expression:]

```
ora:getManager(bpws:getVariableData('inputVariable
```

Task Assignment Evaluation

The assignee is captured as a node set. In the two modes for specifying the assignees, the assignee information is represented as follows:

- Static Assignment

  When the assignment is specified by browsing the user directory, the assignees are visually represented as a comma-separated string. In BPEL, a node set is created with a node created for each user specified.

- Dynamic Assignment

  The XPath expression evaluates to a node set. The value of each node represents a user in the assignee list. There can be only one node in the node set, but the value of each node must be a valid user name. It cannot be a comma-separated string. If the input expression is a comma-separated string, XSLT can be used to create a node set from it.

The assignee list is interpreted in different ways for different types of assignments, as follows:

- Single assignment

The task is assigned to all the assignees in the node set. Each of the assignees can see and approve the task after acquiring it.

- Sequential task

  The task is assigned sequentially to each of the assignees in the node set. For example, if the assignee is set to a node set, as in

  ```
  <user xmlns="ns1-namespace>jstein</user>
  <user xmlns="ns1-namespace>jcooper</user>
  <user xmlns="ns1-namespace>wfaulk</user>
  ```

  then the task is first assigned to `jstein`. On their approval, it is assigned to `jcooper`, and on `jcooper`'s approval, it is assigned to `wfaulk`.

- Parallel task

  In a parallel task, a subtask is created for each node in the node set, and the assignee of each subtask is the value of each node in the node set. For example, if the assignee is set to a node set, then a subtask is created for `jstein`, `jcooper`, and `wfaulk`, as follows:

  ```
  <user xmlns="ns1-namespace>jstein</user>
  <user xmlns="ns1-namespace>jcooper</user>
  <user xmlns="ns1-namespace>wfaulk</user>
  ```

One exception is the sequential task with management chain, where all the assignees specified are treated as a single assignment for the first assignment. The management chain is computed from the previous approver of the task.

The DocumentReview demo demonstrates dynamic assignment from a node set. The `DocumentReview` process is modeled on the parallel task with the final approver pattern.

See: `*Oracle_Home*\integration\orabpel\samples\demos`

Task Assignment Based on External Services

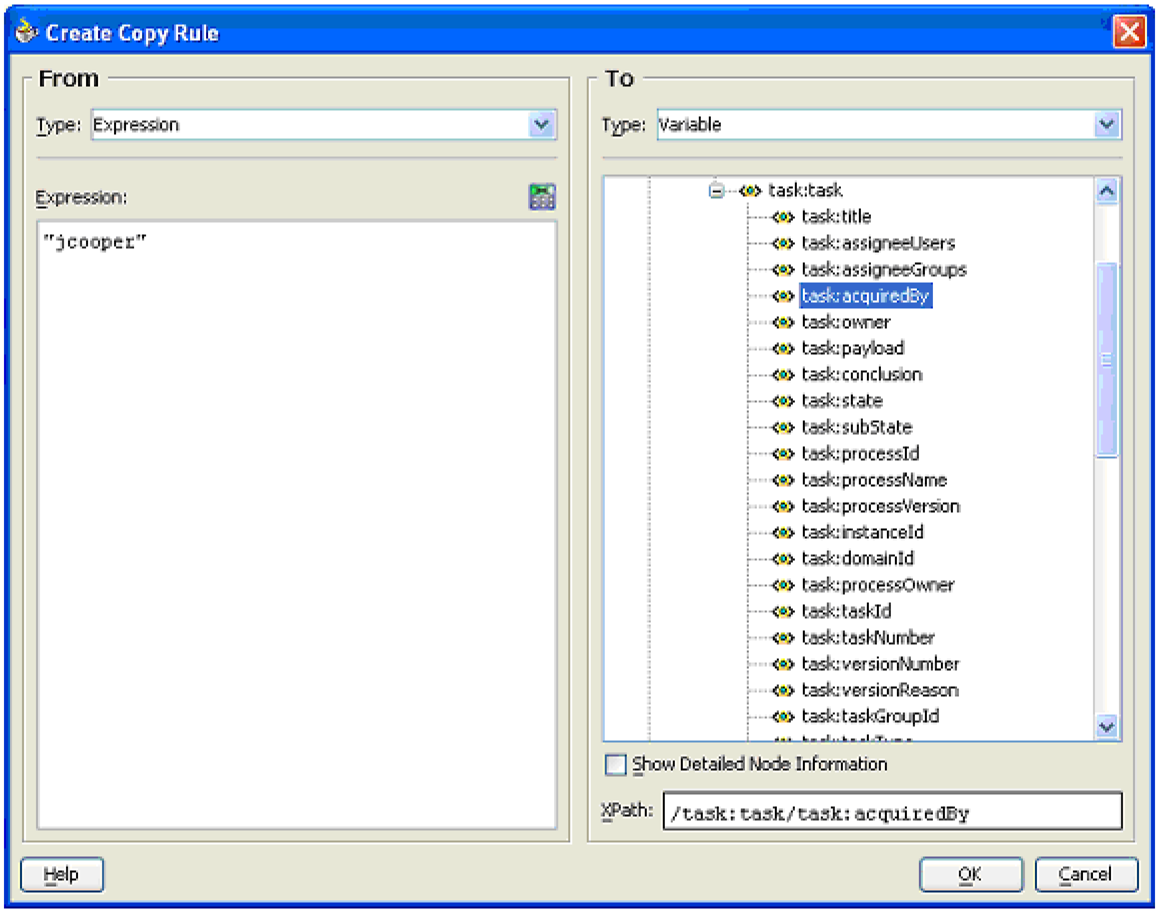
In a scenario where the task assignment is retrieved from an external service, the **Dynamic assignment using XPath expression** option can be used. The element being pointed to by the XPath expression must be either a node (if assigning to a single user or group) or a node set (if assigning to multiple users or groups).

Any complex task assignee computation must be done before the workflow is started. For example, if the assignee of the task should be the one with the least load of all the possible assignees, then a service can be implemented to compute the assignee, and the assignee returned from the service can be set as the task assignee using the XPath option.

Assigning a Task to a Specific User of a Role and Marking It As Acquired

When the task is assigned to a group or multiple users, the task must be acquired before a user can act on the task. It is also possible to set the task as acquired to a user from the business process. This can be done by adding a copy rule in the `assign` activity named `setUserDefinedAttributes`. This copy statement sets the `/task:task/task:acquiredBy` element of the task, as shown in Figure 16–17. The `acquiredBy` value can be computed from a service; for example, a service that does some load balancing among users computes the `acquiredby` value. This return value can be set to the `acquiredBy` element of the task.

Figure 16–17 Copy Rule


Create Copy Rule

From

Type: Expression

Expression:

```
"jcooper"
```

To

Type: Variable

task:task
- task:title
- task:assigneeUsers
- task:assigneeGroups
- task:acquiredBy
- task:owner
- task:payload
- task:conclusion
- task:state
- task:subState
- task:processId
- task:processName
- task:processVersion
- task:instanceId
- task:domainId
- task:processOwner
- task:taskId
- task:taskNumber
- task:versionNumber
- task:versionReason
- task:taskGroupId

Show Detailed Node Information

XPath: /task:task/task:acquiredBy

Setting Task Assignees from a Dynamic Delimited String

In a scenario in which the assignees are represented as a delimited string (for example, a comma-separated string), the delimited string must be converted to a node set to set the task assignees from it. For this purpose, you can use `orcl:create-nodeset-from-delimited-string`. The arguments for this function are as follows:

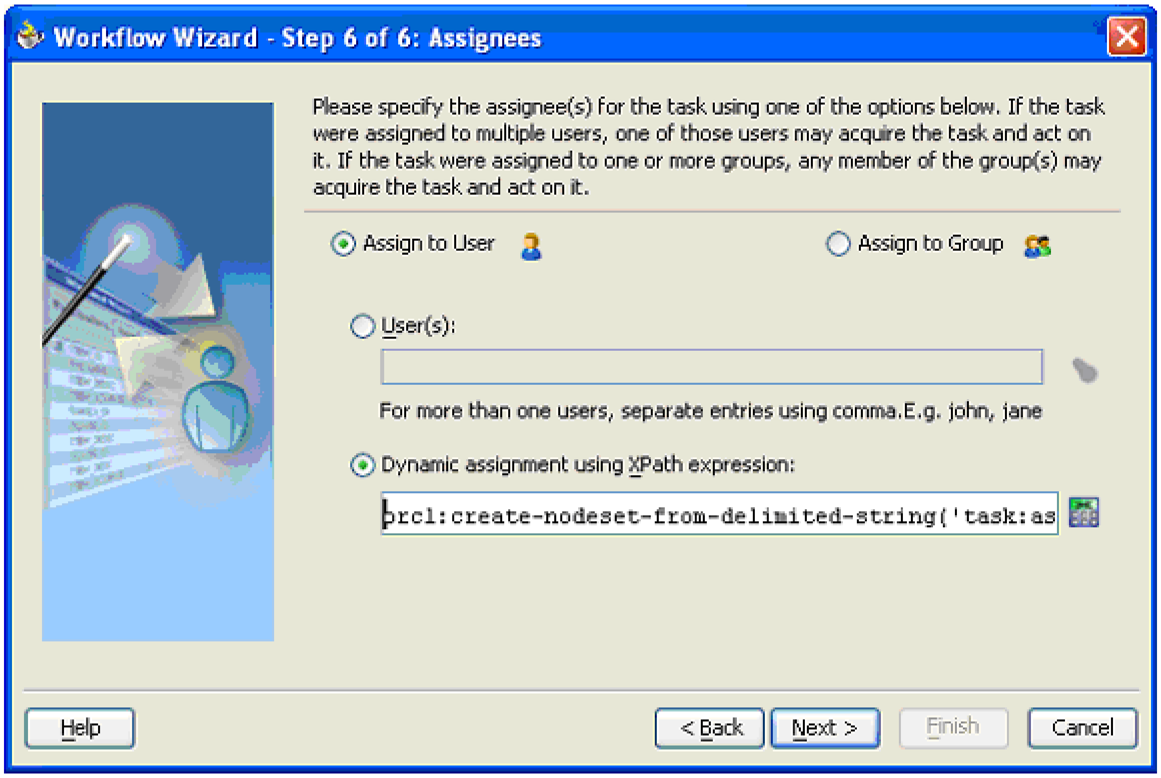
- `QName` in which each node in the node set must be created. The `QName` can be represented in two forms:
  - `task:assignee`
  - `{http://mytask/task}assignee`
- Delimited string

  The delimiter used to delimit the string, which can be `,` (comma) or `;` (semicolon), for example.

For example, the BPEL variable `inputVariable`, part `payload`, and query `/client:TestExtensionFunctionProcessRequest/client:input` represent the assignees in a comma-separated format (example, `jcooper,jstein,wfaulk`) and this comma-separated list contains the assignees of the task. This string cannot be used as-is to set the task assignees because the task assignees are expected to be a node set. To set the assignees, select the dynamic assignment in the task assignees page, as shown in Figure 16–18, and set it to

```
orcl:create-nodeset-from-delimited-string('task:assignee',
bpws:getVariableData('inputVariable','payload',
'/client:TestExtensionFunctionProcessRequest/client:input'), ',')
```

Figure 16–18 Workflow Wizard—Assignees



Please specify the assignee(s) for the task using one of the options below. If the task were assigned to multiple users, one of those users may acquire the task and act on it. If the task were assigned to one or more groups, any member of the group(s) may acquire the task and act on it.

[✓ Assign to User] Assign to Group

User(s):

For more than one users, separate entries using comma.E.g. john, jane

[✓ Dynamic assignment using XPath expression:]

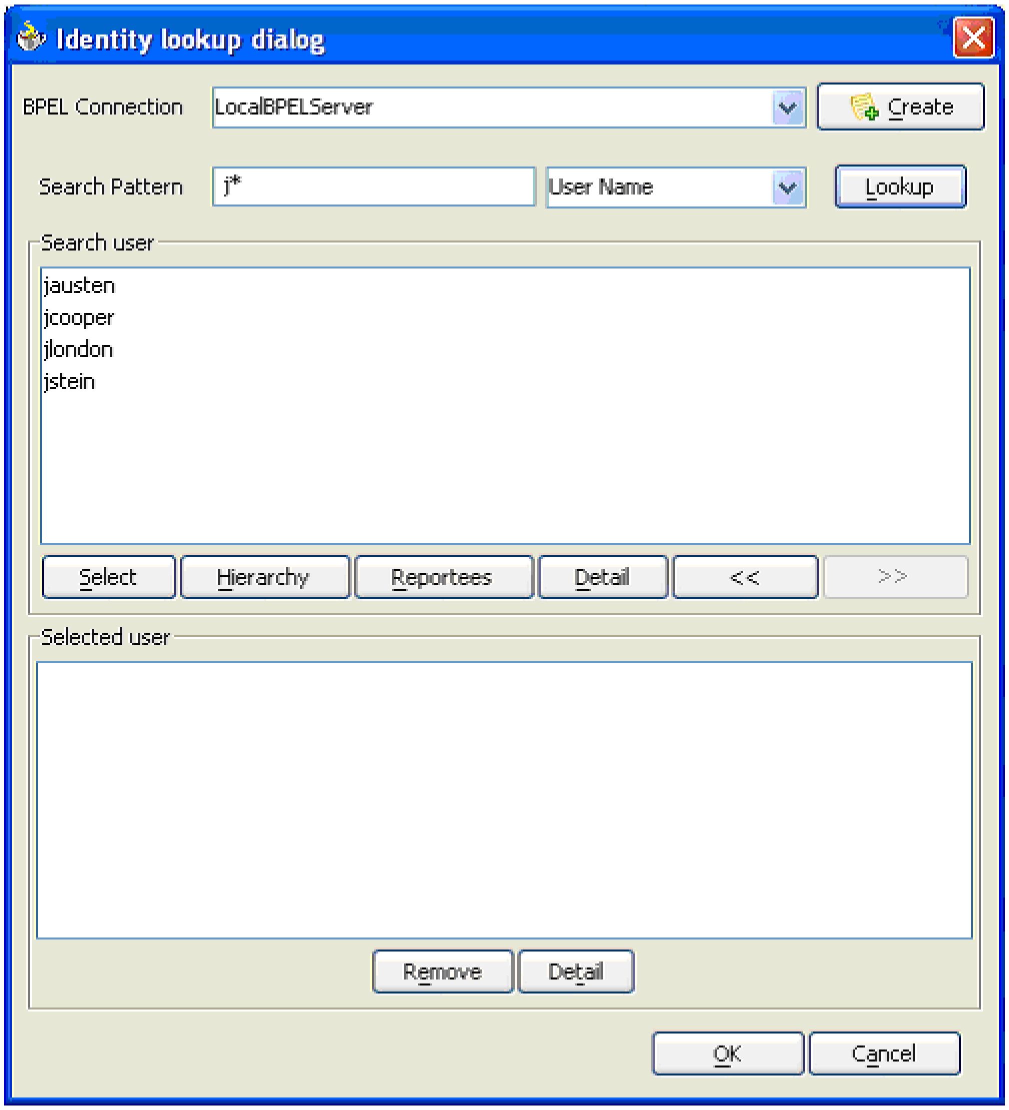
orcl:create-nodeset-from-delimited-string('task:as

Selecting Users or Groups by Browsing the User Directory

Assignees can be selected by browsing the user directory (Oracle Internet Directory (OID), JAZN/XML, LDAP, and so on) that is configured for use by Oracle BPEL Process Manager. Clicking the flashlight icon to browse OID users (next to the **User(s)** or **Group(s)** text boxes) produces the Identity lookup dialog window.

In the identity lookup dialog window, shown in Figure 16–19, you can search by entering a search string such as `jcooper, j*, *`, and so on. Clicking **Lookup** fetches all the users that match the search criteria.

Figure 16–19 Identify Lookup Dialog



BPEL Connection: LocalBPELServer

Search Pattern: j* — User Name

Search user

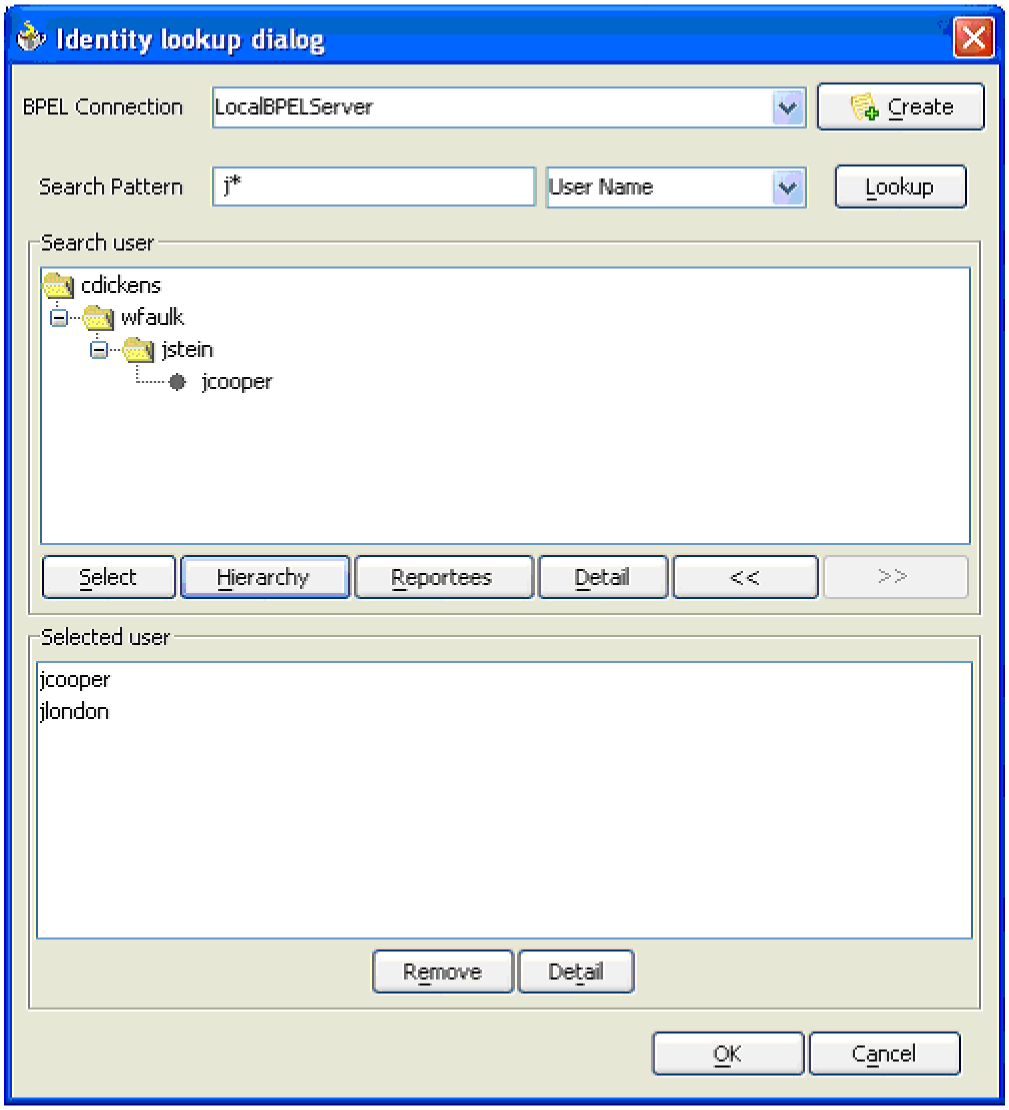
jausten
jcooper
jlondon
jstein

Selected user

One or more users or groups can be highlighted and selected by clicking **Select**.

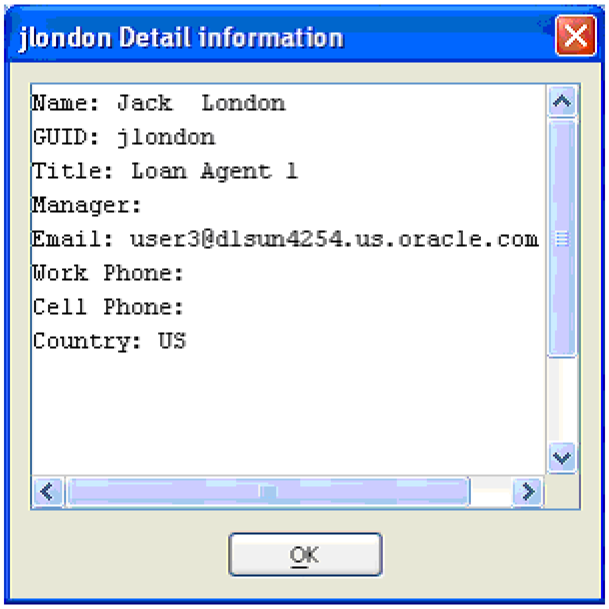
You can view the hierarchy of a user by highlighting the user and clicking **Hierarchy**, as shown in Figure 16–20. Similarly, clicking **Reportees** displays the reportees of a selected user or group.

Figure 16–20 User Hierarchy



You can view the details of a user or group by highlighting the user or group and clicking **Detail**, as shown in Figure 16–21.

Figure 16–21 Detail Information for a User



```
Name: Jack  London
GUID: jlondon
Title: Loan Agent 1
Manager:
Email: user3@dlsun4254.us.oracle.com
Work Phone:
Cell Phone:
Country: US
```

Adding a Task Attachment from a Business Process

You can add task attachments from the BPEL process and the Worklist Application. Task attachments can be one of the following types:

- URI—A URI attachment is a reference to a URL or file location.
- Content—A content attachment file is a file that is uploaded.

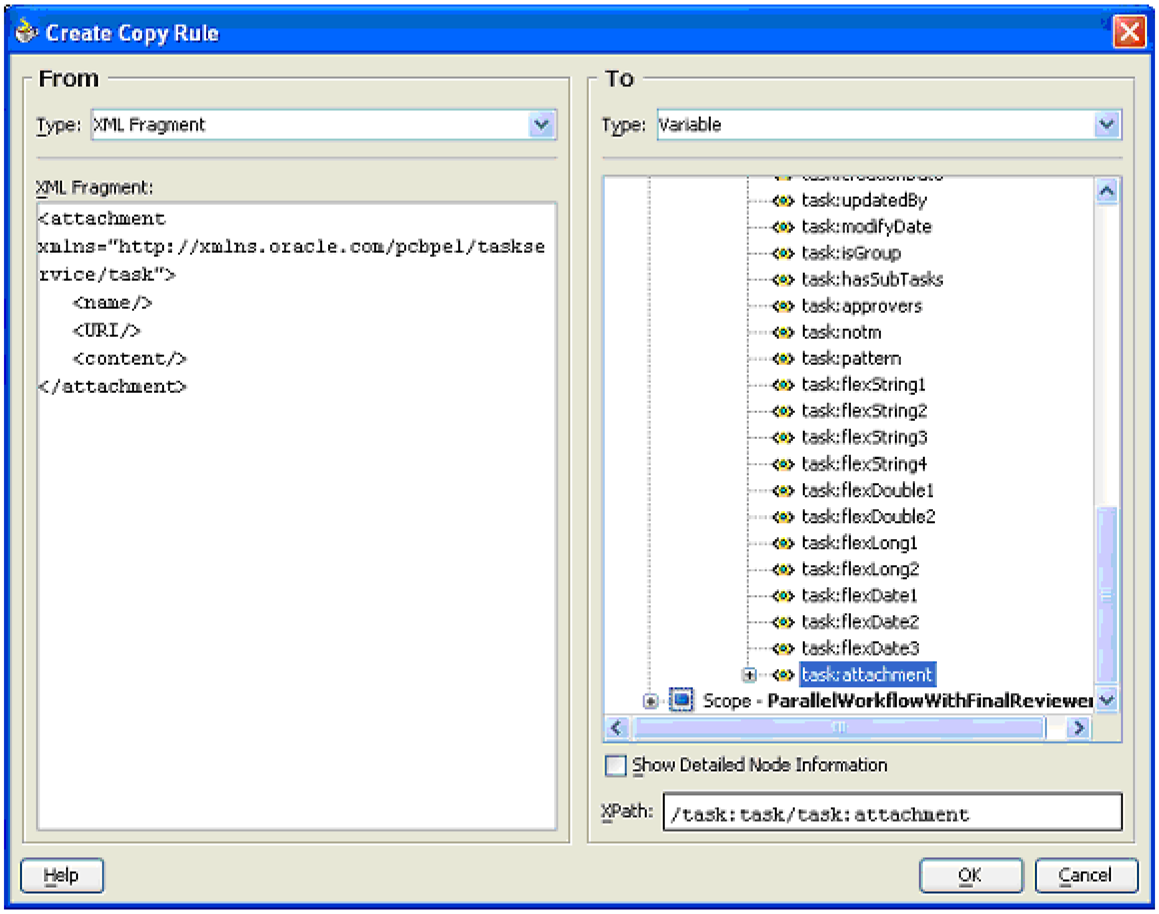
From the BPEL process, you can set attachments on the task variable by adding the following copy statements in the `assign` named `setUserDefinedAttributes` in the workflow scope.

1. In the Create Copy Rule window, set the **From** part to be the following **XML Fragment**:

```
<attachment xmlns="http://xmlns.oracle.com/pcbpel/taskservice/task">
  <name/>
  <URI/>
  <content/>
</attachment>
```

   The attachment element in the task object is initially empty, so the attachment element must be initialized.

2. Select the **To** part to be the attachment element in the task variable.



```
<attachment
xmlns="http://xmlns.oracle.com/pcbpel/taskse
rvice/task">
   <name/>
   <URI/>
   <content/>
</attachment>
```

task:updatedBy
task:modifyDate
task:isGroup
task:hasSubTasks
task:approvers
task:notm
task:pattern
task:flexString1
task:flexString2
task:flexString3
task:flexString4
task:flexDouble1
task:flexDouble2
task:flexLong1
task:flexLong2
task:flexDate1
task:flexDate2
task:flexDate3
task:attachment
Scope - ParallelWorkflowWithFinalReviewer

XPath: `/task:task/task:attachment`

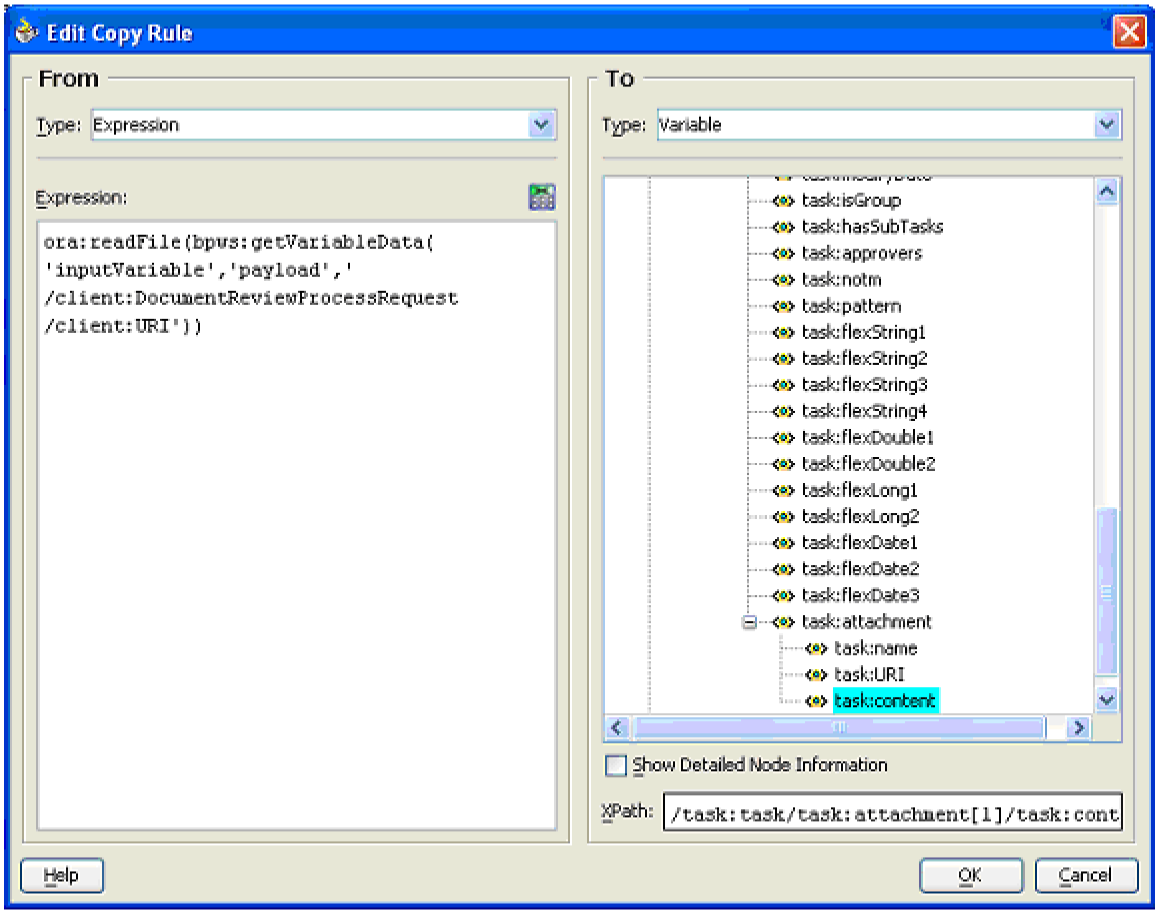
3. Add another copy statement that sets the attachment content. For example, if the attachment is available at a URI captured by the input message, the XPath extension function `ora:readFile` is as follows (this is the **From** expression):

```
ora:readFile(bpws:getVariableData('inputVariable','payload','/client:DocumentRe
viewProcessRequest/client:URI'))
```

4. Select the **To** part as `/task:task/task:attachment[1]/task:content` of the task variable that was created by the wizard.

   By default, the tree does not add the index 1, so it must be added manually.

If the attachment is a URI attachment, the **To** is set to `/task:task/task:attachment[1]/task:URI` of the task variable.



```
ora:readFile(bpws:getVariableData(
'inputVariable','payload','
/client:DocumentReviewProcessRequest
/client:URI'))
```

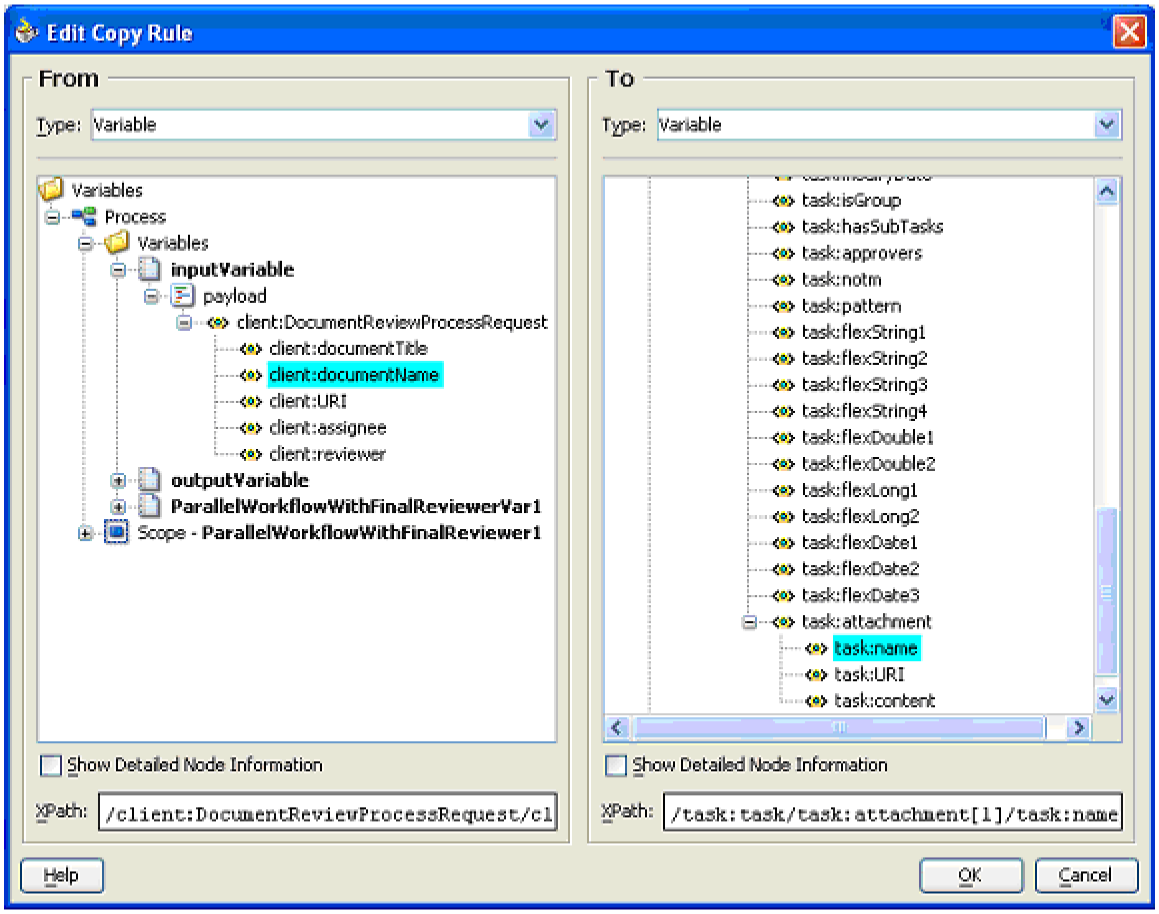
XPath: `/task:task/task:attachment[1]/task:cont`

5. Add another copy statement that sets the attachment name.

   For example, if the name is captured in a business process variable, select that element for the copy **From**.

6. Select the **To** as `/task:task/task:attachment[1]/task:name` of the task variable that was created by the wizard.

By default, the tree does not add the index 1, so it must be added manually.



From — Type: Variable

XPath: /client:DocumentReviewProcessRequest/cl

To — Type: Variable

XPath: /task:task/task:attachment[1]/task:name

The DocumentReview demo demonstrates adding task attachments from the BPEL process. The document review process sends a document to multiple users for their review. The document to be reviewed is set as a task attachment.

See: *Oracle_Home*`\integration\orabpel\samples\demos`

Actions Performed on a Task

The Worklist Application enables users to participate in a BPEL process by performing tasks that require manual intervention. The worklist user interface displays tasks specific to the logged-in user based on the user's permissions and assigned groups and roles. The types of actions that the user 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. Additionally, a user can modify flex fields, task priority, or include comments or 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 their worklist and routed to the next approver or back to the business process based on the workflow pattern.
- 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. The various system actions allowed on a task are as follows:

– Escalate—This operation enables a user to escalate a task to their manager for further action.

– Reassign—A manager can delegate a task to reportees. Similarly, the process owner or a user with BPMWorkflowReassign privileges can delegate a specific task to any other person in the organization.

– Request More Information—Any participant in the workflow can request more information from the task creator 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.

– Request More Information with Reapproval—Any participant in the workflow can request more information from the task creator or any of the prior approvers of the task and require the approvers who have approved the task reapprove it. This action is supported only if the current user task supports sequential approval. 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 when `jcooper` submits the information, `jstein` and `wfaulk` approve the task before it comes to `cdickens`. If `cdickens` requests information with reapproval from `jstein` and then `jstein` submits the information, then `wfaulk` approves the task before it comes to `cdickens`.

– Submit More Information—This operation enables a user to respond to a request for additional information. This action is performed after the user has made the necessary updates to the task or has added comments or attachments containing additional information.

– Route—This operation 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.

– Suspend—This operation enables process 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).

– Resume—This operation enables process 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.

– Acquire—This operation enables a user to obtain an exclusive right to work on a task that is assigned to a group or multiple users. No action can be performed on a task assigned to a group or multiple users until it is acquired. Only one user can acquire a task at any given time.

– Release—This operation enables a user to abandon the exclusive right to work on a task that is assigned to a group or multiple users. After a task is released, any other user who is assigned to the task can acquire it.

– 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 modeler has restricted task renewal on the workflow.

– Withdraw—The creator of the task can withdraw any pending task if they are no longer interested in sending it further through the workflow. A process owner can also withdraw a task on behalf of the creator. When a task is withdrawn, the business process is called back with the state attribute of the task set to `Withdrawn`.

See Also:

- "Identity Service" on page 16-75 for details on the roles in the system
- Chapter 17, "Worklist Application" for information on how to perform the system actions

Simple Workflow

Simple workflow is used when a task must be acted on by one user. The task can be assigned to a set of users or groups. If the task is assigned to multiple participants, one of them must acquire the task and complete it. See "Actions Performed on a Task" on page 16-29 for a description of the actions a user can perform from the Worklist Application.

Figure 16–22 shows how simple workflow is implemented in BPEL.

Figure 16–22 A Simple Workflow Implemented in Oracle BPEL Process Manager

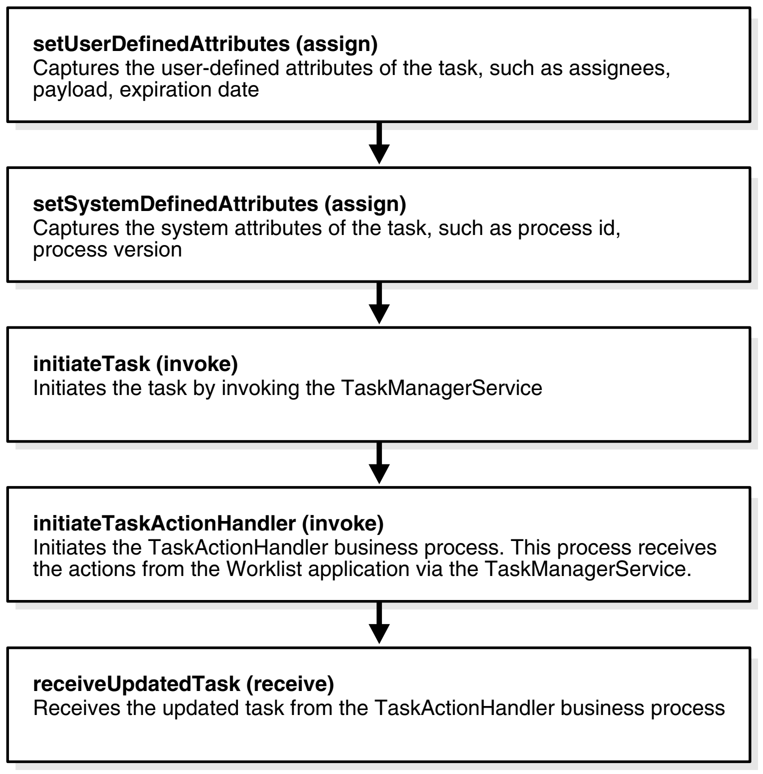
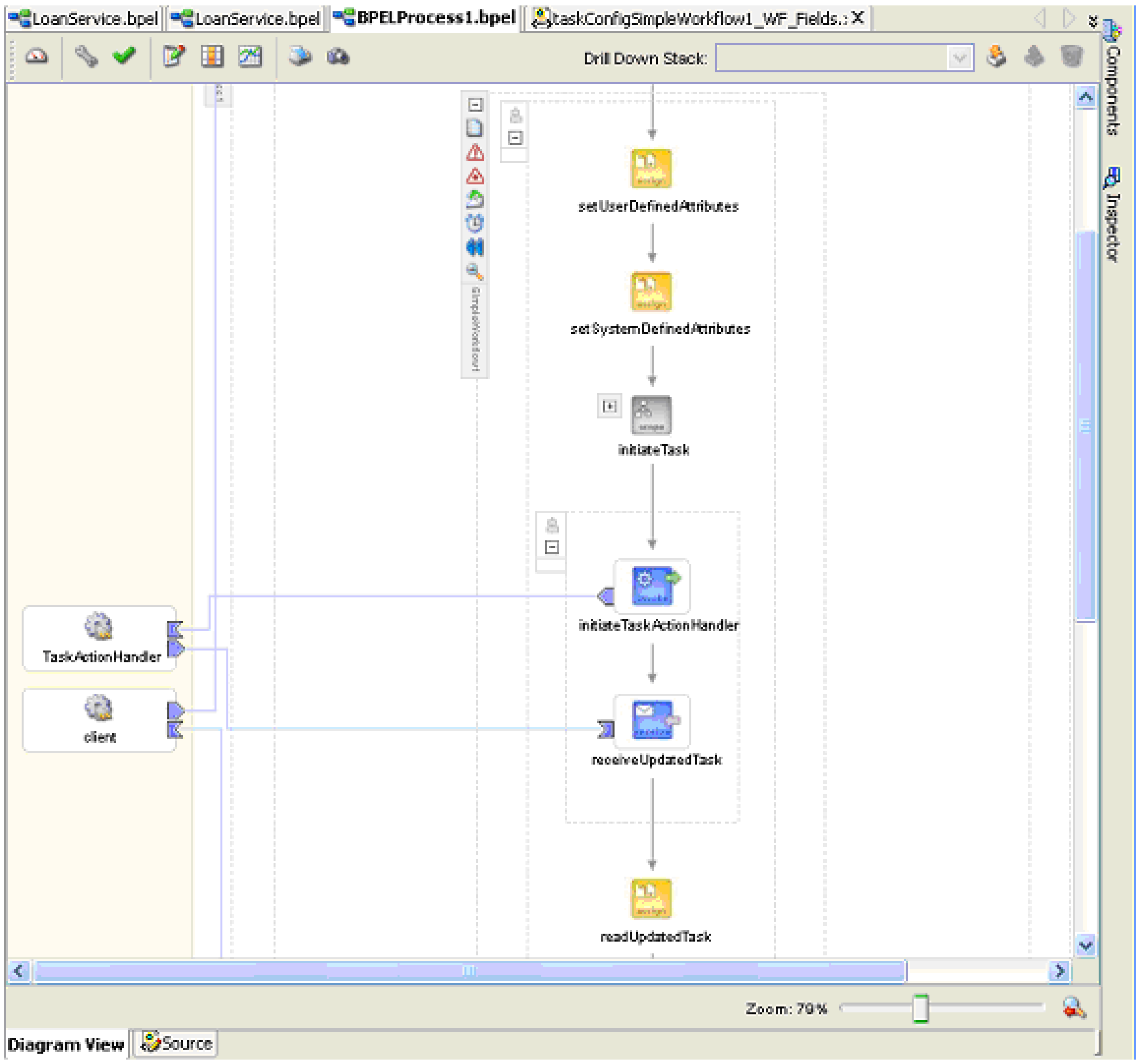

setUserDefinedAttributes (assign)
Captures the user-defined attributes of the task, such as assignees, payload, expiration date
setSystemDefinedAttributes (assign)
Captures the system attributes of the task, such as process id, process version
initiateTask (invoke)
Initiates the task by invoking the TaskManagerService
initiateTaskActionHandler (invoke)
Initiates the TaskActionHandler business process. This process receives the actions from the Worklist application via the TaskManagerService.
receiveUpdatedTask (receive)
Receives the updated task from the TaskActionHandler business process

Figure 16–23 shows a user task modeled using the simple workflow in JDeveloper BPEL Designer.

Figure 16–23 A Simple Workflow Modeled in JDeveloper BPEL Designer



Use Case

An employee, through the employee portal, submits a vacation request. The portal initiates a business process that includes a user task modeled using a simple workflow. The task is assigned to the manager of the employee. When the manager approves or rejects the vacation request, the employee is notified with the manager's decision by e-mail. See the VacationRequest example in the samples directory for an implementation of this use case.

Customizations for Simple Workflow

The following customizations are possible:

- Changing the assignee after creating the user task

  The `assign` named `setUserDefinedAttributes` in the generated scope captures the setting of the task assignee. Depending on whether the task is assigned to a user or a group, the task attribute `assigneeUsers` or `assigneeGroups` is set. This assignment can be changed to modify the task assignee. For example, if you are using an external service that does load balancing or determines task assignee based on contents of the payload, you can call this service and set the assignees to a local variable. This variable can then be used to do a dynamic assignment based on an XPath expression.

- Changing the outcomes after creating the user task

  The outcomes are captured in the XML task configuration file, where they can be changed manually. See "Task Outcomes" on page 16-13 for more information.

- Changing the user task to support multiple approvals

  A simple workflow (or any variation of simple workflow) does not support multiple approval because it is intended for a single approval. If reapproval is required, change the user task to sequential workflow.

Simple Workflow with Automatic Escalation

Simple workflow with automatic escalation is a variation of the simple workflow in which the task is escalated when the task expires. The rules of escalation are specified as part of creating the user task. As in a simple workflow, the task can be assigned to a set of users or groups. See "Actions Performed on a Task" on page 16-29 for a description of the actions a user can perform from the Worklist Application. If the user does not perform any of the custom actions before the task expires, the task is escalated to their manager as specified in the user directory that is configured with the identity service.

Figure 16–24 shows how simple workflow with automatic escalation is implemented in Oracle BPEL Process Manager.

Figure 16–24 Simple Workflow with Automatic Escalation

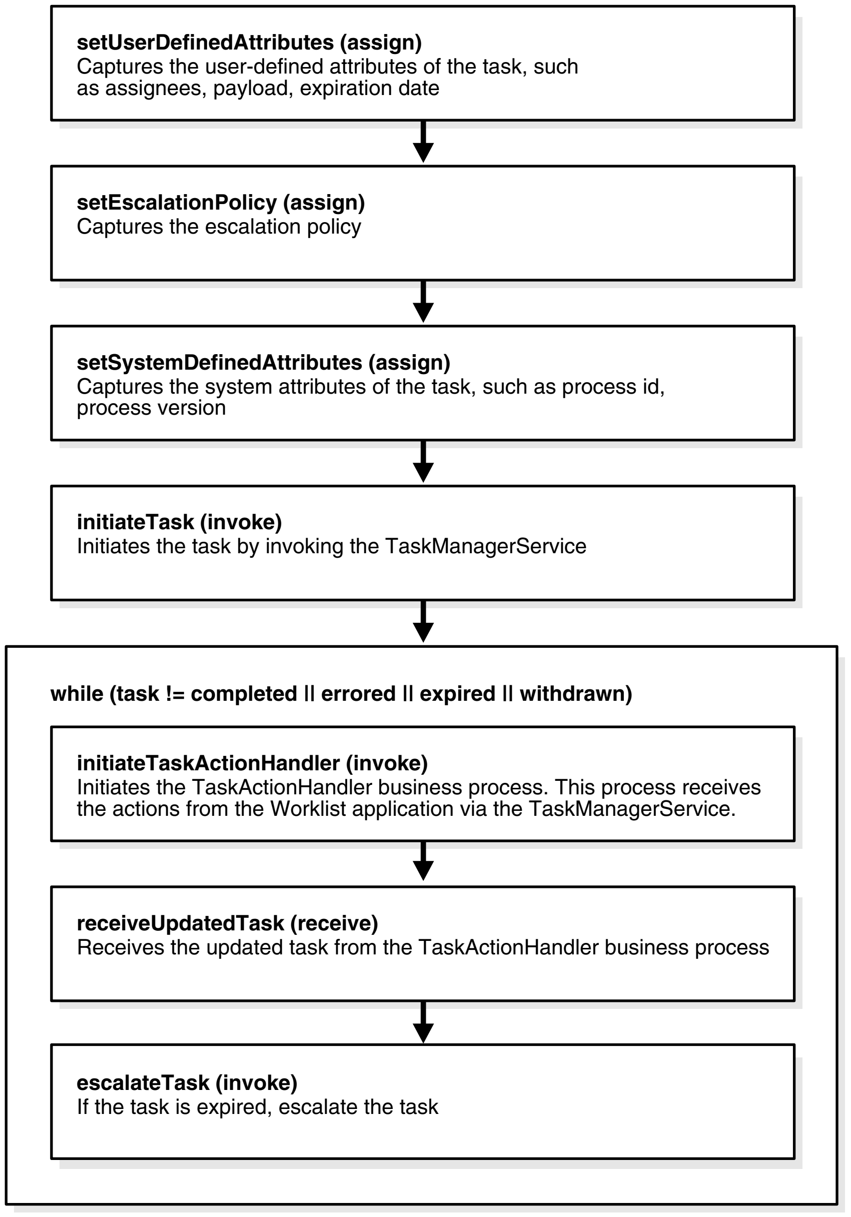
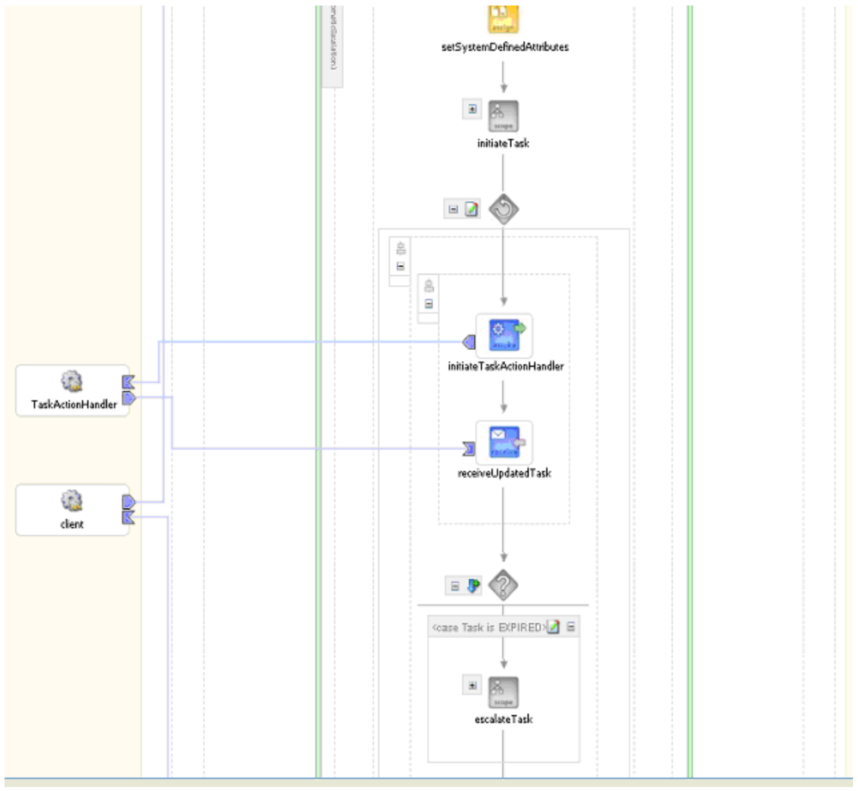

setUserDefinedAttributes (assign)
Captures the user-defined attributes of the task, such as assignees, payload, expiration date
setEscalationPolicy (assign)
Captures the escalation policy
setSystemDefinedAttributes (assign)
Captures the system attributes of the task, such as process id, process version
initiateTask (invoke)
Initiates the task by invoking the TaskManagerService
while (task != completed || errored || expired || withdrawn)
initiateTaskActionHandler (invoke)
Initiates the TaskActionHandler business process. This process receives the actions from the Worklist application via the TaskManagerService.
receiveUpdatedTask (receive)
Receives the updated task from the TaskActionHandler business process
escalateTask (invoke)
If the task is expired, escalate the task

Figure 16–25 shows the user task scope modeled using the simple workflow with automatic escalation.

Figure 16–25 Simple Workflow with Automatic Escalation in JDeveloper BPEL Designer



Use Case

The HelpDeskServiceRequest process gives users the ability to file help desk service request tickets. If the person who receives the ticket does not act on it within a specified time period, the ticket is automatically escalated to their manager. The ticket is automatically escalated three times if no one has acted on it within a predefined time, until it gets to the CEO of the company. If the CEO also does not act on it, it expires.

Pattern-Specific Parameters

Two parameters are specific to this pattern.

- The maximum number of times the task can be escalated

  This required parameter specifies how many times the task is escalated. For example, if this parameter to set to `2`, then the task is assigned to the user jcooper. If none of the users acts on the task in time, the task is assigned to jcooper (initial assignee), user jstein (jcooper's manager) and user wfaulk (jstein's manager).

- The title of a user to whom the task can be escalated

  This optional parameter specifies the title of the last user to whom the task can be escalated. For example, if this parameter is set to **Manager**, the task is assigned to user jcooper. If none of the users acts on the task in time, the task is assigned to jcooper (initial assignee) and user jstein (jcooper's manager), whose title is **Manager**.

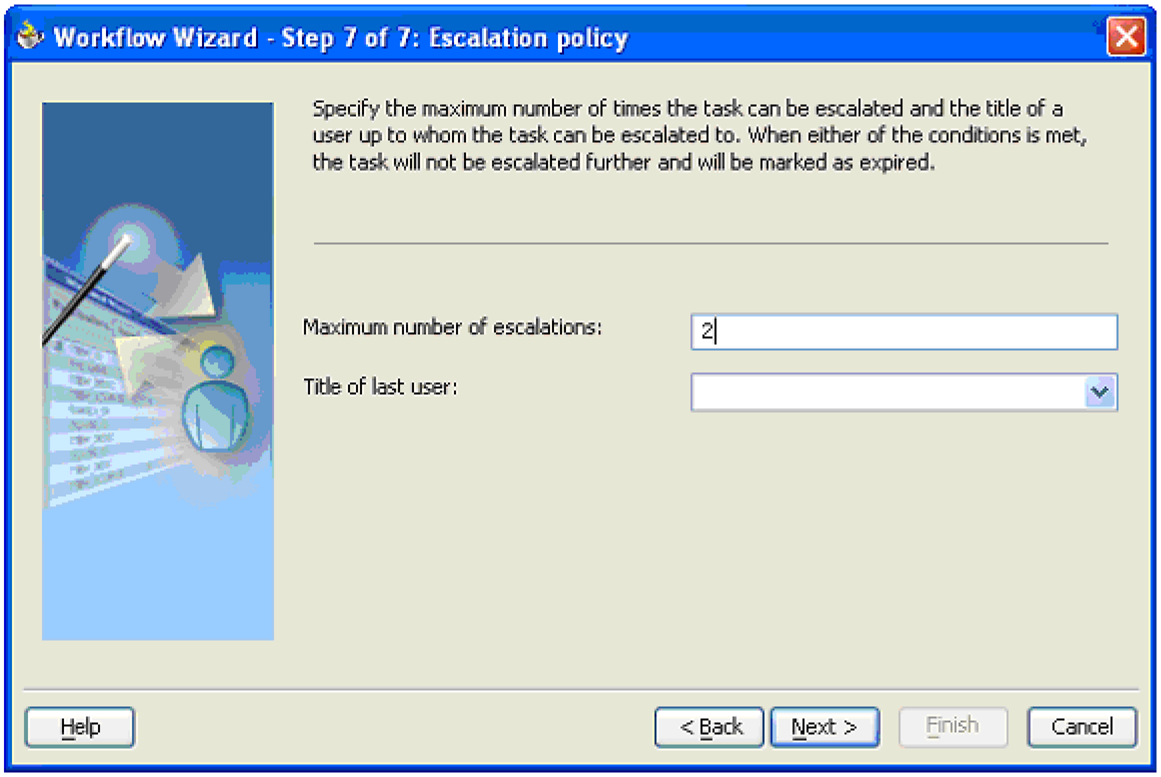
  The title can be typed in or selected from a list of prespecified titles. The list of titles that are displayed is configured in `defaultUserTitles.xml` at

  `Oracle_Home``\integration\jdev\jdev\system10.1.2.0.0.1811\`

When both these parameters are specified, the task is escalated until one of the conditions causes the escalation to stop. For example, if the maximum number of times is **4** and the title is **Manager**, then the task is assigned to jcooper. If none of the users acts on the task in time, the task is assigned to jcooper (initial assignee), and user jstein (jcooper's manager), whose title is **Manager**.

Figure 16–26 shows where these parameters are specified.

Figure 16–26 Escalation



Specify the maximum number of times the task can be escalated and the title of a user up to whom the task can be escalated to. When either of the conditions is met, the task will not be escalated further and will be marked as expired.

Maximum number of escalations: 2

Title of last user:

Customizations for Simple Workflow with Automatic Escalation

The customizations in "Customizations for Simple Workflow" on page 16-32 apply to this pattern, in addition to the following.

- Changing the duration for the task assignment

  When the task is expired and is escalated, the task is renewed for a duration equal to the initial expiration duration of the task. The BPEL `assign` named `setEscalationPolicy` in the user task scope sets the renewal duration in the variable `oraBPMExpDuration`. This copy statement can be changed to change the expiration duration for the subsequent task assignments.

- Changing the number of levels of escalation

  The BPEL `assign` named `setEscalationPolicy` in the user task scope captures the number of levels of escalation in the variable `oraBPMManagementChain` (the query for the copy is `/identityservice:managementChain/identityservice:levels`). This can be changed to any appropriate value.

- Changing the title of the last user to whom the task is escalated

The BPEL `assign` named `setEscalationPolicy` in the user task scope stores the title of the last user in the variable `oraBPMManagementChain` (the query for the copy is `/identityservice:managementChain/identityservice:uptoTitle`). This can be changed to any appropriate value.

- Changing the user to whom the task is escalated on task expiration

  By default, on task expiration, the task is escalated to the user's manager. This can be changed by changing the generated BPEL scope. In the generated BPEL scope, there is another BPEL scope named `escalateTask` that performs the escalation. In the scope, change the `escalateTask` invoke to invoke the `updateTask` operation on the `TaskManagerService` instead of the `escalateTask` operation. The `TaskManagerService.wsdl` file defines the `updateTask` operation. The task assignees (users or groups) to whom the task is escalated must be set before the operation is invoked. In addition, the state of the task (`task:task/task:state`) must be set to `ASSIGNED` before the operation is invoked.

Simple Workflow with Automatic Renewal

Simple workflow with automatic renewal is a variation of the simple workflow in which the task is renewed when the task expires. The rules of renewal are specified as part of creating the user task. As in a simple workflow, the task can be assigned to a set of users or groups. See "Actions Performed on a Task" on page 16-29 for a description of the actions a user can perform from the Worklist Application. If the user does not perform any of the custom actions before the task expires, the task is renewed for a specific duration.

Figure 16–27 shows how simple workflow with automatic renewal is implemented in Oracle BPEL Process Manager.

Figure 16–27 Simple Workflow with Automatic Renewal

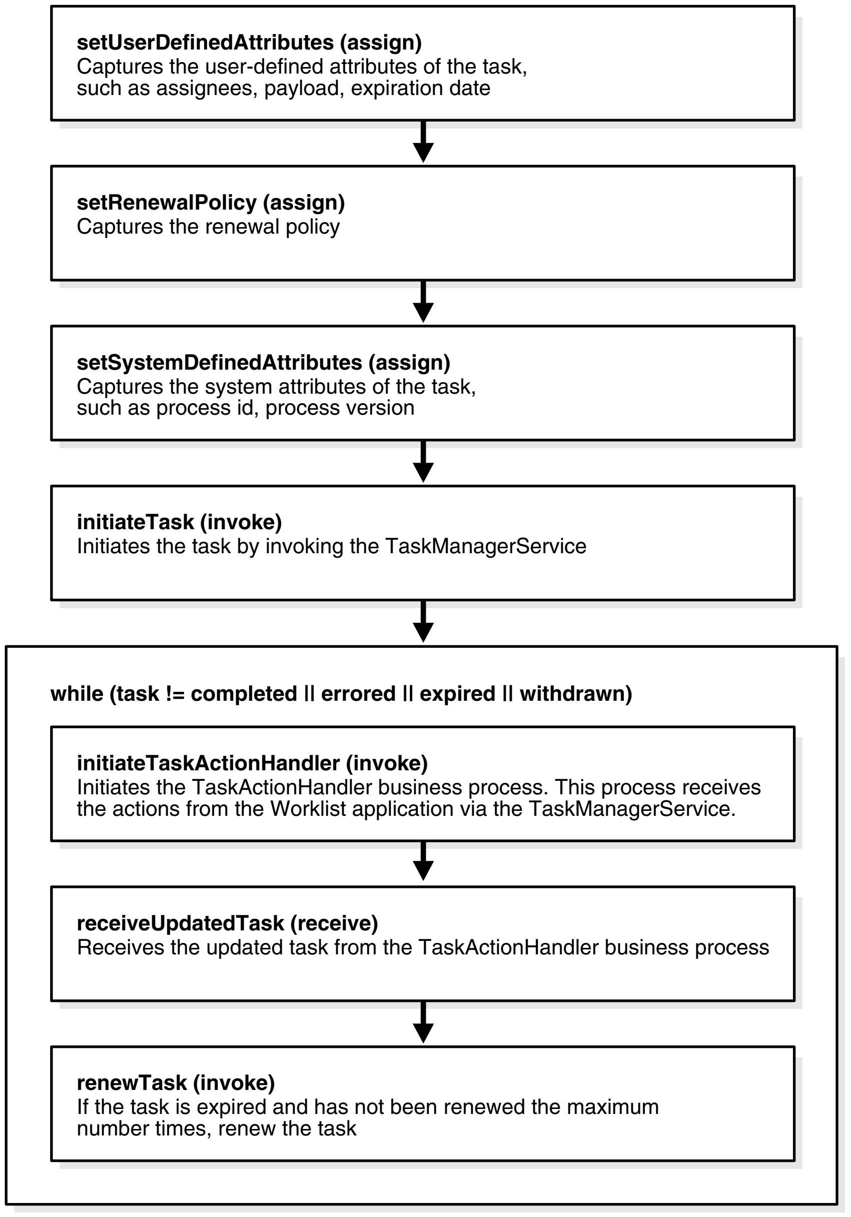
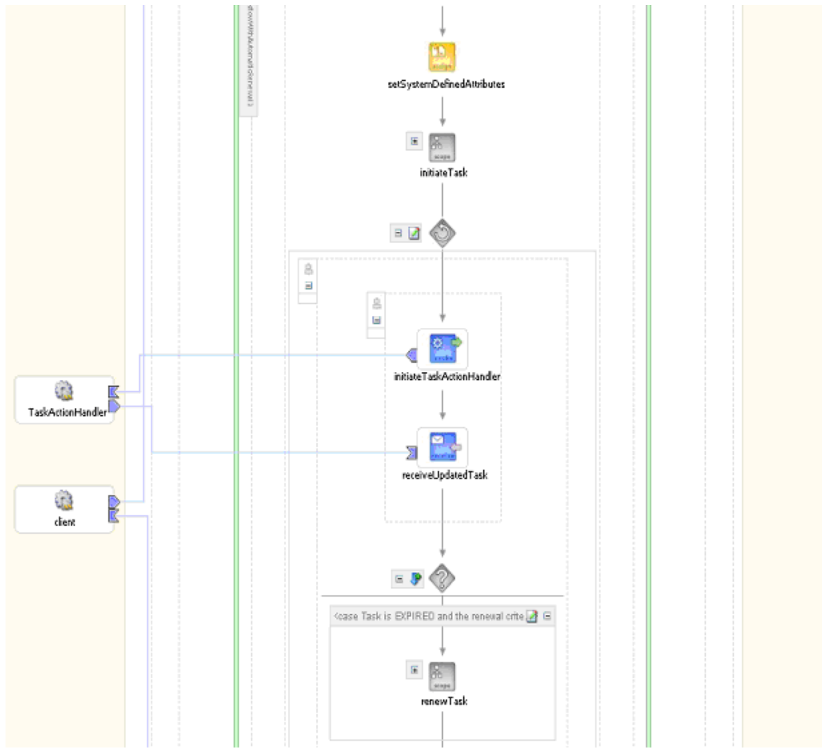

setUserDefinedAttributes (assign)
Captures the user-defined attributes of the task, such as assignees, payload, expiration date
setRenewalPolicy (assign)
Captures the renewal policy
setSystemDefinedAttributes (assign)
Captures the system attributes of the task, such as process id, process version
initiateTask (invoke)
Initiates the task by invoking the TaskManagerService
while (task != completed || errored || expired || withdrawn)
initiateTaskActionHandler (invoke)
Initiates the TaskActionHandler business process. This process receives the actions from the Worklist application via the TaskManagerService.
receiveUpdatedTask (receive)
Receives the updated task from the TaskActionHandler business process
renewTask (invoke)
If the task is expired and has not been renewed the maximum number times, renew the task

Figure 16–28 shows the user task scope modeled using the simple workflow with automatic renewal.

Figure 16–28 Simple Workflow with Automatic Renewal


setSystemDefinedAttributes
initiateTask
initiateTaskActionHandler
TaskActionHandler
receiveUpdatedTask
client
<case Task is EXPIRED and the renewal crite
renewTask

Use Case

A business process manages renewal of magazine subscriptions. The rules of the subscription are that if the user cancels the subscription one month before the expiration date of the current subscription, then the subscription is cancelled free of charge. If the user cancels in the last month of the current subscription, then the subscription is cancelled with a charge. If the user does not cancel, it is renewed.

The process is initiated 60 days before the subscription is up for renewal. The process contains a user task modeled with the simple workflow with automatic renewal pattern. When the magazine subscription task is assigned to a user, an e-mail is sent to the user. The user can go to a portal and renew or cancel the subscription. Depending on the user action, the business process determines the cancellation or cancellation with charge or renewal.

Pattern-Specific Parameters

One parameter is specific to this pattern.

- The maximum number of times the task can be renewed

  This required parameter specifies how many times the task is renewed.

Customizations for Simple Workflow with Automatic Renewal

The customizations in "Customizations for Simple Workflow" on page 16-32 apply to this pattern, in addition to the following:

- Changing the duration for the task assignment

  When the task expires, it is renewed. The renewal duration of the task is the same as it was for the first task assignment. The BPEL `assign` named `setRenewalPolicy` in the user task scope sets the new expiration duration in the variable `oraBPMRenewDuration`. This copy statement can be changed to change the expiration duration for the subsequent task assignments.

- Changing the number of levels of renewal

  The BPEL `assign` named `setRenewalPolicy` in the user task scope stores the maximum number of levels of renewal in the variable `oraBPMMaxTimesRenewed`. This can be changed to any appropriate value.

- Adding logic depending on number of times renewed

  The BPEL scope level variable `oraBPMNumOfTimesRenewed` captures the number of times the task is renewed. This variable can be used to build additional logic in the business process.

Sequential Workflow

Sequential workflow is used when a task must be approved by multiple users. The users or groups to whom the task is assigned can be specified in one of the following ways:

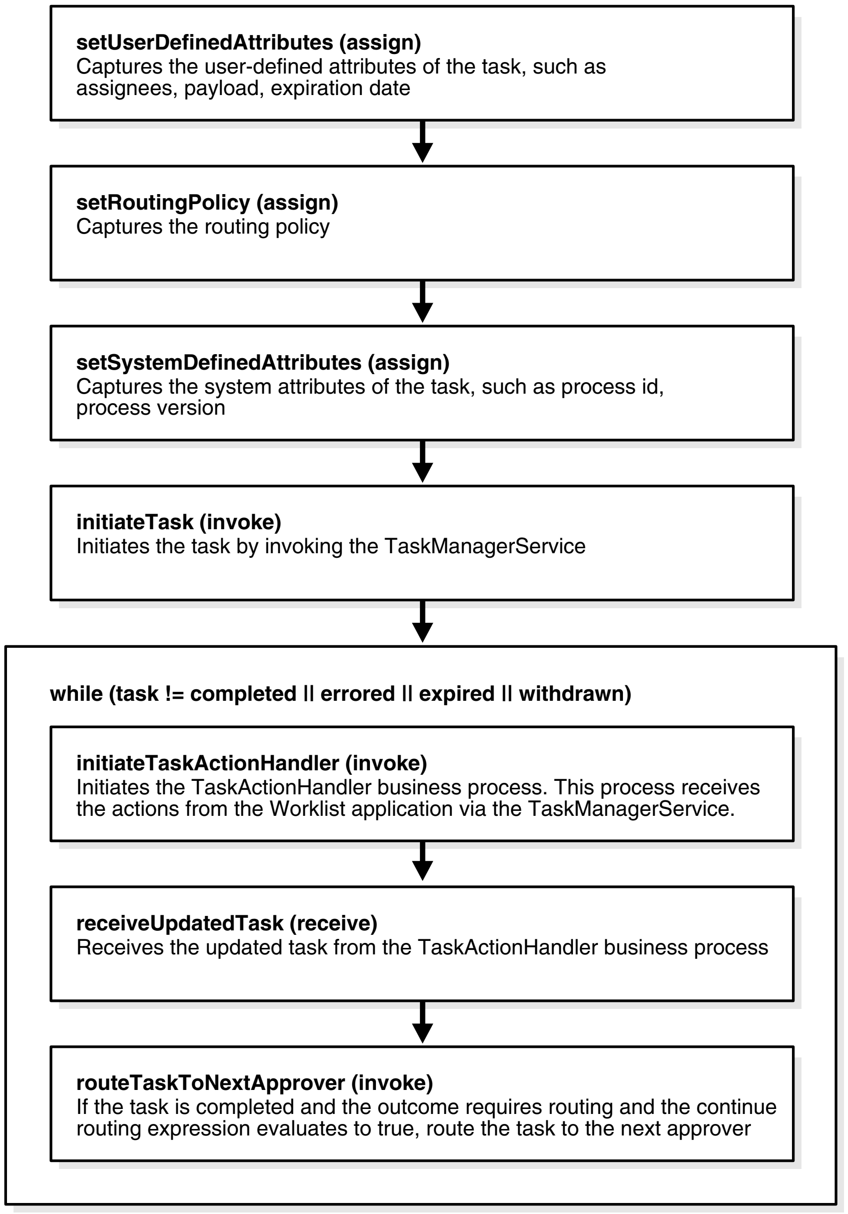
- List of users in a management chain
- List of users or groups from the identity service set during the design time of the process
- Dynamic list of users or group from another business process variable

In each of the preceding cases, the task is sequentially assigned to each of the users in the list. See "Actions Performed on a Task" on page 16-29 for a description of the actions a user can perform from the Worklist Application. In a sequential workflow, a request for information with reapproval is permitted.

As part of modeling a sequential workflow, the list of outcomes that cause the task to be routed can also be specified. For example, if a task has two outcomes, `ACCEPT` and `REJECT`, the business logic might require the task to be routed to the next approver only if the current user accepts the task. A continue routing expression can also be specified. If this condition evaluates to true, the task is routed.

Figure 16–29 shows how the sequential workflow is implemented in Oracle BPEL Process Manager.

Figure 16–29 How Sequential Workflow Is Implemented in BPEL


setUserDefinedAttributes (assign)
Captures the user-defined attributes of the task, such as assignees, payload, expiration date
setRoutingPolicy (assign)
Captures the routing policy
setSystemDefinedAttributes (assign)
Captures the system attributes of the task, such as process id, process version
initiateTask (invoke)
Initiates the task by invoking the TaskManagerService
while (task != completed || errored || expired || withdrawn)
initiateTaskActionHandler (invoke)
Initiates the TaskActionHandler business process. This process receives the actions from the Worklist application via the TaskManagerService.
receiveUpdatedTask (receive)
Receives the updated task from the TaskActionHandler business process
routeTaskToNextApprover (invoke)
If the task is completed and the outcome requires routing and the continue routing expression evaluates to true, route the task to the next approver

Based on the assignees, a future approver function is created to capture future approvers of the task. This function is persisted in the approvers element of the task. If the management chain assignment is used, the `managementChain` function is created. If a list of users is provided (either using static assignment from a user directory or an XPath function), a list function is created. The list function has a user or group function in it to represent the assignees. This approver function creation is done in the `assign` named `setRoutingPolicy` in the generated workflow scope. This function is not changed in normal circumstances; however, it can be changed if a complicated sequential assignment is being created. For example, first assign to users `jcooper` and `jstein`, then assign to `wfaulk`, and so on. (The complexity of this example is multiple users assignment—`jstein` and `jcooper`—for the first assignment).

Examples of approver functions are as follows:

- `managementChain("2", "Manager")`

Routes the task to users in the management chain. The management chain includes users within two levels and up to a user whose title is Manager.

- `users("jcooper", "jstein")`

  Routes the task once to users `jcooper` and `jstein`.

- `users("jcooper", "jstein", acquiredBy("jcooper"))`

  Routes the task once to users `jcooper` and `jstein`. Also sets the `acquiredBy` to `jcooper`.

- `groups("LoanAgentRole", "Supervisor", acquiredBy("jcooper"))`

  Routes the task once to groups `LoanAgentRole` and `Supervisor`. Sets the acquired-by to `jcooper`.

- `list(users("jcooper", "jstein"), groups("LoanAgentRole"), acquiredBy("jcooper"))`

  Routes the task once to users `jcooper` and `jstein` and group `LoanAgentRole` and also sets the `acquiredBy` to `jcooper`.

- `list(users("jcooper","jstein")), list(groups("LoanAgentRole"))`

  Routes the task to users `jcooper` and `jstein`. When one of those users acquires and acts on the task, routes the task to group `LoanAgentRole`.

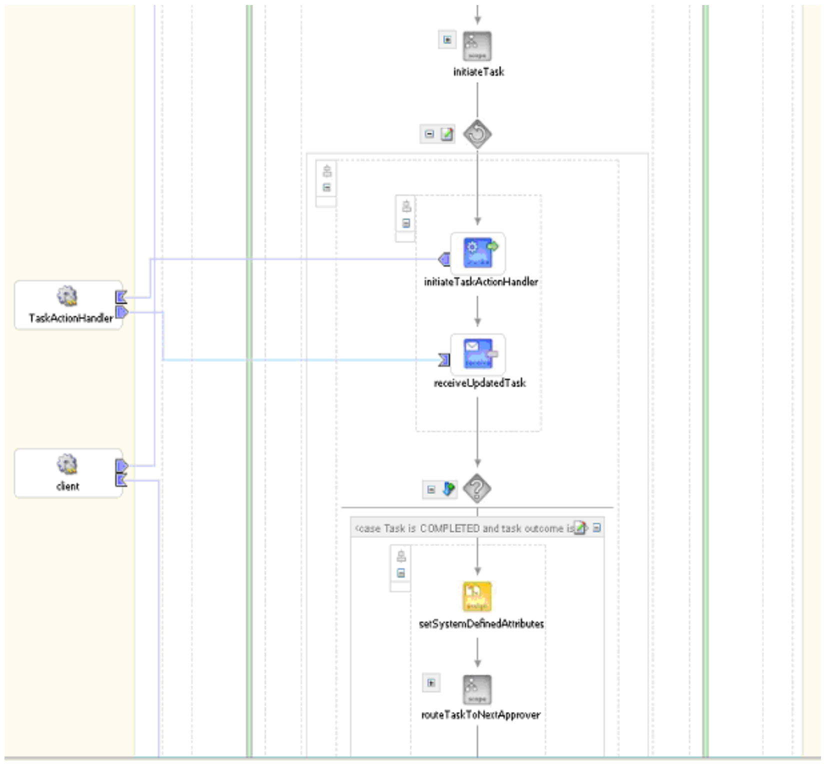
- `adhoc()`

  The task supports adhoc routing and the next users or groups are specified by the current approver of the task.

These functions are evaluated when the task must be routed to the next approver. See "Approver Functions" on page 16-95 for more information, and for how to modify the value of the approvers element to achieve different results.

Figure 16–30 shows the user task scope modeled using the sequential workflow.

Figure 16–30 Sequential Workflow in JDeveloper BPEL Designer


initiateTask
TaskActionHandler
initiateTaskActionHandler
receiveUpdatedTask
client
<case Task is COMPLETED and task outcome is
setSystemDefinedAttributes
routeTaskToNextApprover

Use Cases

A loan application processing system receives loan applications. These loan applications are initially assigned a group called LoanAgentRole. A user in this group evaluates and approves the loan application and provides a loan offer. If the loan application amount is greater than 100,000 US dollars, then the loan application and the initial loan offer are send to the initial approver's manager. After they approve it, the loan application is routed to their manager. If the loan application amount is less than or equal to 100,000 US dollars, then the initial loan offer is sent back to the caller. This use case is demonstrated in the LoanDemoPlusWithWorkflow sample.

Another use case is when a purchase order approval system processes a purchase order using a business process. An employee belonging to a group named Supervisor initially evaluates the purchase order. After the initial user approves the purchase order, their manager approves it. After the manager approves the purchase order, the purchase order is forwarded to the billing and shipping departments. This use case is demonstrated in the Order Booking tutorial purchase order approval service.

See Also: *Oracle BPEL Process Manager Order Booking Tutorial*

Pattern-Specific Parameters

The sequential workflow has the following pattern-specific parameters.

- The approvers assignment policy
- Outcome that results in the task being routed
- Continue routing expression

In simple workflow, the assignees are specified for the only assignment of the task. In sequential workflow, the set of users to whom the task is routed is specified. There are two ways to specify this.

1. A list of users from a business process variable or a static list set at design time.
2. A management chain. This involves selecting the initial assignees (a set of groups or users) and parameters to select the management chain. The parameters to select the management chain include specifying the following attributes:
    a. The number of levels in the management chain

       This **required** parameter specifies how many levels in the management chain are included in the list. For example, if this parameter to set to 2 and the task is initially assigned to a user `jcooper`, then both the user `jstein` (`jcooper`'s manager) and user `wfaulk` (`jstein`'s manager) are included in the list, apart from `jcooper` (the initial assignee).
    b. The title of the last user in the management chain

       This **optional** parameter specifies the title of the last user in the management chain. For example, if this parameter is set to 'Manager' and the task is assigned to a user `jcooper`, user `jstein` (`jcooper`'s manager) is included in the list, apart from `jcooper` (the initial assignee).

       The title can be typed in or selected from a list of prespecified titles. The list of titles is configured in `defaultUserTitles.xml` at

       ```
       Oracle_Home\integration\jdev\jdev\system10.1.2.0.0.1811\
       ```

When both these parameters are specified, then the task routing is determined by both the title and the number of levels. The routing continues until one of the conditions—title or the number of levels—no longer applies. The following examples illustrate this.

Example: If the maximum level in the management chain is four and the title is 'Manager' and the task is assigned to a user `jcooper`, then the list includes `jcooper` (initial assignee) and user `jstein` (`jcooper`'s manager), whose title is 'Manager'. The task is not be routed beyond the "Manager", even though the maximum number of levels is set to `4`. This is useful in cases where you want the approvals to go up to a certain management level (for example, Director).

Example: If the maximum level in the management chain is 1 and the title is 'Director' and the task is assigned to a user `jcooper`, then the list includes `jcooper` (initial assignee) and user `jstein` (`jcooper`'s manager, whose title is 'Manager'). It does not include `wfaulk` (`jstein`'s manager, whose title is 'Director') because the task was already routed to one user beyond the initial assignee.

There are also two parameters that determine if the task must be routed.

1. The list of outcomes that cause the task to be routed can also be specified. For example, if a task has two outcomes, ACCEPT and REJECT, the business logic might require the task to be routed to the next approver only if the current user accepted the task.
2. A continue routing expression can also be specified. If this expression evaluates to true, the task is routed further after a specific user acts on it. This expression can be used to dynamically control how many approvals are required. Examples of this expression are as follows:
    a. The task is routed only if the loan amount is greater than `1000`; otherwise it is approved only once.

```
bpws:getVariableData('loan',
'/auto:loan/auto:loanApplication/auto:loanAmount') > number(10000)
```

b. The task is routed until it is approved by a manager.

```
ora:getPreviousApproversTitle('/task:task/task:taskId') = string('Manager')
```

c. The task is approved at most three times.

```
ora:getNumberOfTimesApproved('/task:task/task:taskId') < 3
```

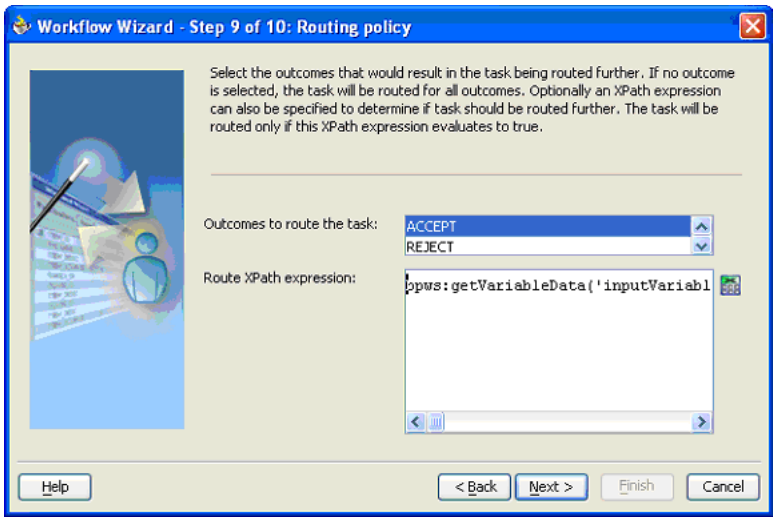
d. The task is approved at most three times when the loan amount is greater than `10000`; otherwise it should be approved only once.

```
ora:getNumberOfTimesApproved('/task:task/task:taskId') < 3 and
bpws:getVariableData('loan',
'/auto:loan/auto:loanApplication/auto:loanAmount') > number(10000)
```

The extension function `ora:getPreviousApproversTitle()` gets the title of the previous task approver. The function `getNumberOfTimesApproved()` gets the number of times a task was approved. See "Workflow-Related XPath Extension Functions" on page 16-89 for a list of the extension functions.

Figure 16–31 shows the routing parameters.

Figure 16–31 Routing Parameters



Select the outcomes that would result in the task being routed further. If no outcome is selected, the task will be routed for all outcomes. Optionally an XPath expression can also be specified to determine if task should be routed further. The task will be routed only if this XPath expression evaluates to true.

Outcomes to route the task: ACCEPT, REJECT

Route XPath expression: bpws:getVariableData('inputVariabl

Customizations for Sequential Workflow

- Changing the initial assignee in the management chain after creating the user task

  The `assign` named `setUserDefinedAttributes` in the generated scope captures the assignment of the task assignee. Depending on whether the task was assigned to a user or a group, the task attribute `assigneeUsers` or `assigneeGroups` is set. This assignment can be changed to modify the task assignee.

- Changing the management chain parameters after creating the user task

  The `assign` named `setRoutingPolicy` in the generated scope sets the approver function in the task attribute `approvers`. The approver function contains the management chain parameters. The parameters can be changed in this function. See "Approver Functions" on page 16-95 for more information.

- Changing the assignees in the list after creating the user task

  The `assign` named `setUserDefinedAttributes` in the generated scope captures the assignment of the task assignee. The assignment to the `oraBPMRouteAssignees` variable attributes `/taskmngr:assigneeEntities/taskmngr:assignee` captures the list of users or groups.

- Changing the outcomes after creation of the user task

  Task outcomes are captured in the XML task configuration file, where outcomes can be changed manually. See "Task Outcomes" on page 16-13 for more information.

- Changing routing policy

  The `assign` named `setRoutingPolicy` captures the routing policy. The future assignees of a task in a sequential workflow are captured in the approvers attribute of the task object. This assign sets the approvers attribute to an approver function depending on the assignment policy chosen—management chain or list of users. See "Approver Functions" on page 16-95 for more information.

- Changing continue routing expression

  In the generated scope, the continue routing expression is evaluated in the `assign` named `evaluateRoutingCriteria`. The condition that is set in the wizard is negated with a `not()` in this expression. This expression can be changed as needed.

- Adding continue routing expression

  The response from the each user is received in the `receive` activity `receiveUpdatedTask`. The `receive` activity follows a `switch` activity in which one of the case statements is "Task is COMPLETED and the task outcome is one of the outcomes specified in the routing policy". The expression on this case statement can be changed to add any continue routing expression.

- Adding approvers

  In the management chain routing policy, approvers can be added by changing the management chain parameters as described previously. Similarly, by changing the assignees in the list of users as described previously, approvers can be added.

- Changing the user or group to whom the task is routed based on output of an external system.

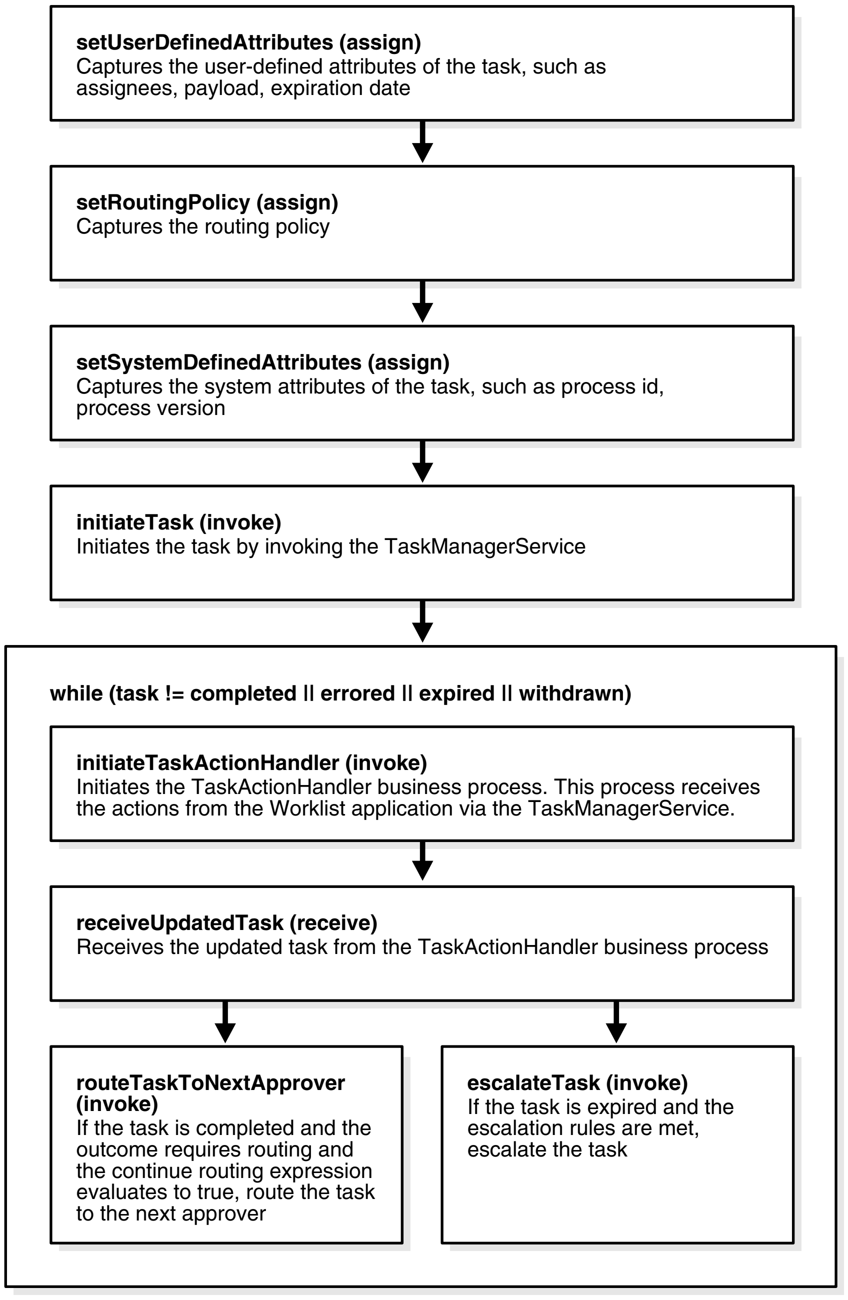
  In the generated BPEL, there is a scope named `routeTaskToNextApprover`. This scope is responsible for routing the task to the next user or group. By default, this routing is based on the approver function. In a scenario where the next task assignee is determined by an external system, the activities in the `routeTaskToNextApprover` scope can be modified to achieve that. The TaskRoutingService exposes an operation called `routeTask` that routes the task to a specified user or group. The input message for this operation includes the task, an id that identifies either a user or a group, and a Boolean flag that identifies the entity as a group or a user. After a message is created, change the `routeTaskToNextApprover` invoke and its input variable accordingly to use the `routeTask` operation instead of the `routeTaskToNextApprover` operation.

Sequential Workflow with Escalation

The sequential workflow with escalation is used when a task must be approved by multiple users and the task must be escalated when it expires. This pattern is an extension of the sequential workflow and all that applies to the sequential workflow also applies to this pattern.

Figure 16–32 shows how sequential workflow is implemented in Oracle BPEL Process Manager.

Figure 16–32 Sequential Workflow with Escalation



setUserDefinedAttributes (assign)
Captures the user-defined attributes of the task, such as assignees, payload, expiration date
setRoutingPolicy (assign)
Captures the routing policy
setSystemDefinedAttributes (assign)
Captures the system attributes of the task, such as process id, process version
initiateTask (invoke)
Initiates the task by invoking the TaskManagerService
while (task != completed || errored || expired || withdrawn)
initiateTaskActionHandler (invoke)
Initiates the TaskActionHandler business process. This process receives the actions from the Worklist application via the TaskManagerService.
receiveUpdatedTask (receive)
Receives the updated task from the TaskActionHandler business process
routeTaskToNextApprover (invoke)
If the task is completed and the outcome requires routing and the continue routing expression evaluates to true, route the task to the next approver
escalateTask (invoke)
If the task is expired and the escalation rules are met, escalate the task

Use Case

An employee (for example, `jcooper`) requests a new laptop urgently. The sequential workflow first routes the task to their manager for approval (for example, `jstein`) and then to a person in the procurement department (for example, `jlondon`) to acquire the laptop. If the workflow is set up with automatic escalation after two days, then the request first goes to the manager for approval and if it is not approved in two days, then the task can be automatically routed to the next person in the management hierarchy (`jstein`'s manager, `wfaulk`). Similar escalation can also be done for the procurement representative. This enables the workflow to complete without long delays in the approval process.

Pattern-Specific Parameters

All the parameters that apply to the sequential workflow apply to this pattern as well. In addition to those parameters, the following parameters are specific to this pattern.

- The maximum number of times the task can be escalated

  This required parameter specifies how many times the task is escalated for each assignment. For example, if this parameter to set to `2`, the task is assigned to user `jcooper`. If `jcooper` does not act on the task in time, then the task is escalated to user `jstein` (`jcooper`'s manager). If `jstein` does not act on the task, then the task is escalated to user `wfaulk` (`jstein`'s manager).

- The title of a user up to whom the task can be escalated

  This optional parameter specifies the title of the last user to whom the task can be escalated. For example, if this parameter is set to Manager, then the task is assigned to a user `jcooper`. If none of the users act on the task in time, then the task is assigned to `jcooper` (initial assignee) and user `jstein` (`jcooper`'s manager), whose title is Manager.

  The title can be typed in or selected from a list of prespecified titles. The list of titles is configured in `defaultUserTitles.xml` at

  ```
  Oracle_Home\integration\jdev\jdev\system10.1.2.0.0.1811\
  ```

When both these parameters are specified, the task is escalated until one of the conditions causes the escalation to stop. For example, if the maximum number of times is `4`, and the title is Manager, then the task is assigned to user `jcooper`. If none of the users act on the task in time, then the task is assigned to `jcooper` (initial assignee), user `jstein` (`jcooper`'s manager), whose title is Manager.

Customizations for Sequential Workflow with Escalation

All the customizations that apply to the sequential workflow apply to this pattern also.

- Changing the escalation policy

  The `assign` named `setRoutingPolicy` in the generated scope sets the escalation policy. The title of the last user to escalate up to is set to the attribute `/identityservice:managementChain/identityservice:uptoTitle` of the variable `oraBPMManagementChain`. The level of escalation is set to the attribute `/identityservice:managementChain/identityservice:levels` of the variable `oraBPMManagementChain`. These assignments can be changed as required.

- Changing the expiration duration of the task when escalated

  By default, the expiration duration of the task when escalated is the initial expiration duration of the task. Changing the assignment to the variable `oraBPMExpDuration` in the `assign` `setRoutingPolicy` changes the expiration duration when the task is escalated.

- Changing the user to whom the task is escalated on task expiration

By default, on task expiration, the task is escalated to the user's manager. This can be changed by changing the generated BPEL scope. In the generated BPEL scope, another BPEL scope named `escalateTask` performs the escalation. In the scope, change the `escalateTask invoke` to invoke the `updateTask` operation on the TaskManagerService instead of the `escalateTask` operation. The `TaskManagerService.wsdl` defines the `updateTask` operation. The task assignees (users or groups) to whom the task is escalated must be set before the operation is invoked. In addition, the state of the task (`task:task/task:state`) should be set to `ASSIGNED` before the operation is invoked.

Parallel Workflow

Parallel workflow is used when a task must be approved by multiple users or groups simultaneously. Each user views a subtask, which is a clone of the task at the time of creation. Each user can add attachments and comments independent of the other users. The users reviewing the task in parallel are not able to see the tasks of the other users. This includes the comments, attachments, and outcomes of the other tasks. The owner of the task can see all the subtasks. When the creator withdraws the parent task, all the subtasks are withdrawn as well.

The users and groups that review the task can be specified in one of the following ways. In each of the preceding cases, a subtask is created for each of the users or groups in the list.

- List of users or groups from OID set during the process design time
- List of users or groups from another business process variable

The outcome of the final task is selected based on the outcomes of each of the subtasks. The outcome of the task is selected based on the following criteria:

- Percentage an outcome requires for it to be selected as the final outcome of the task. For example, assume there are two possible outcomes, ACCEPT and REJECT, and there are five subtasks. If two of the subtasks are accepted and three are rejected, and the acceptance percentage required for an outcome is 50%, then the outcome of the task is rejected.
- If none of the outcomes have the required percentage, a default outcome can be specified as the outcome of the task.

Parallel workflow can be configured for early completion. When early completion is specified, that is, when the outcome of the task can be computed with the outcomes of the completed subtasks, then the pending subtasks are withdrawn. If no early completion is specified, the workflow waits for all the responses.

Figure 16–33 shows how parallel workflow is implemented in Oracle BPEL Process Manager.

Figure 16–33 Parallel Workflow

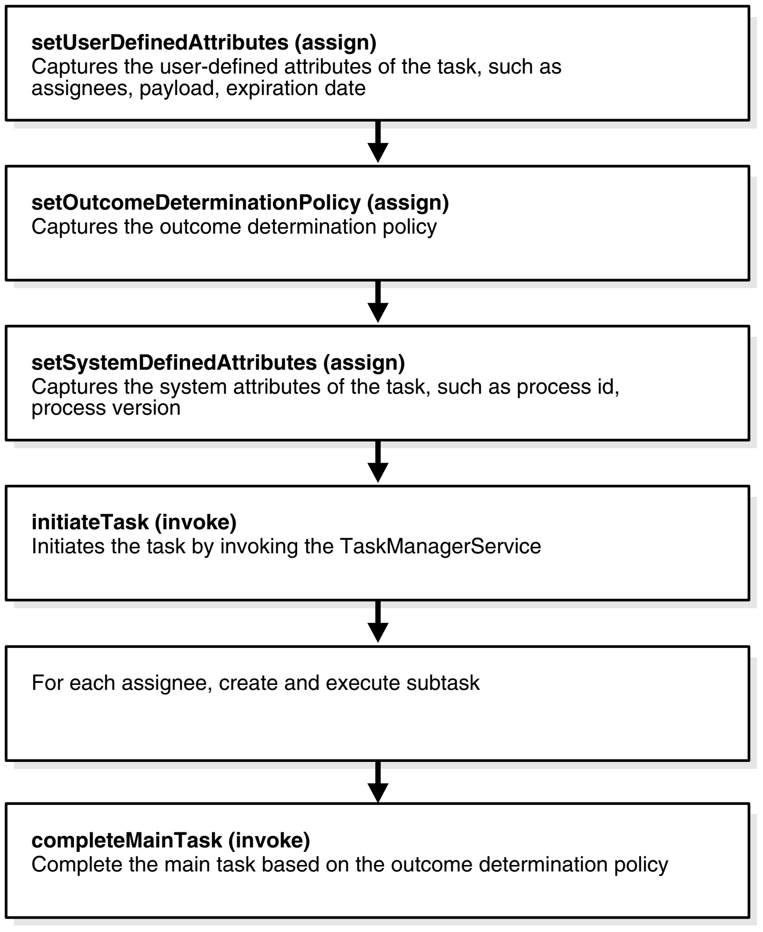

setUserDefinedAttributes (assign)
Captures the user-defined attributes of the task, such as assignees, payload, expiration date
setOutcomeDeterminationPolicy (assign)
Captures the outcome determination policy
setSystemDefinedAttributes (assign)
Captures the system attributes of the task, such as process id, process version
initiateTask (invoke)
Initiates the task by invoking the TaskManagerService
For each assignee, create and execute subtask
completeMainTask (invoke)
Complete the main task based on the outcome determination policy

Figure 16–34 shows the subtask execution process.

Figure 16–34 The Parallel Workflow Subtask Process

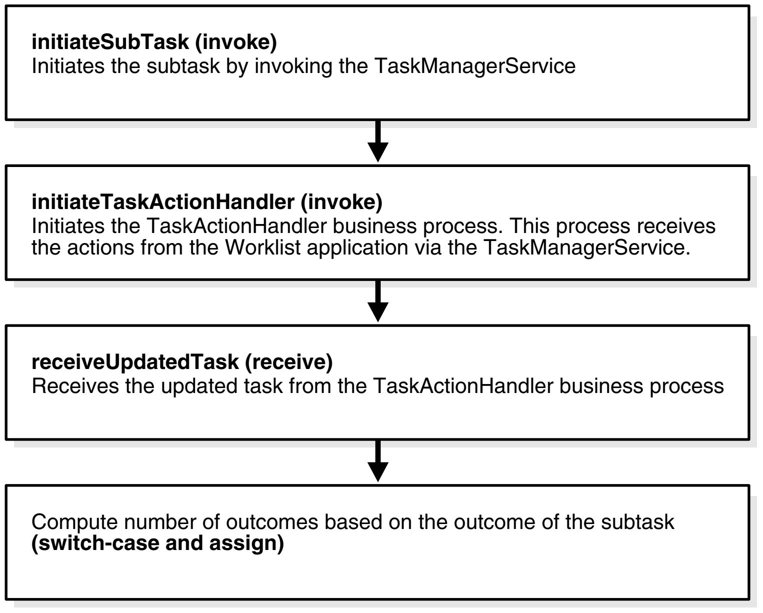
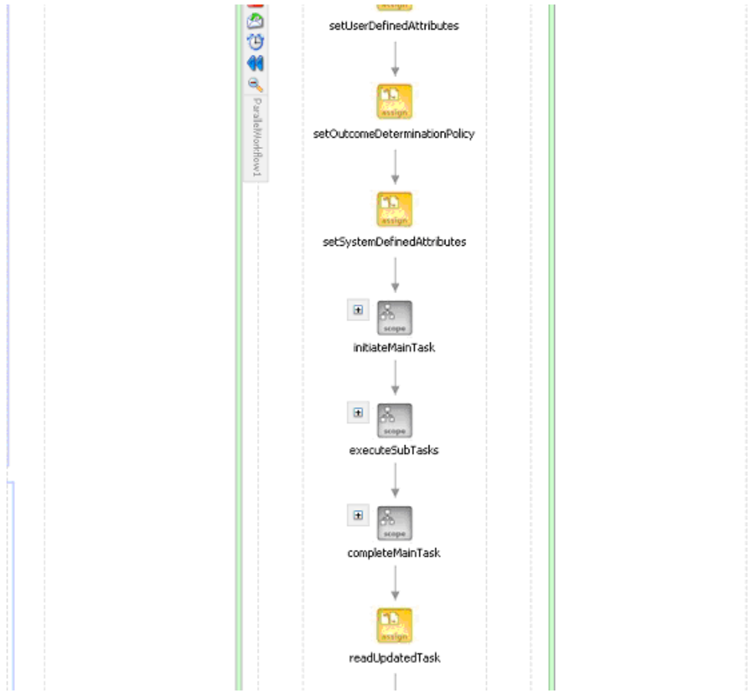

initiateSubTask (invoke)
Initiates the subtask by invoking the TaskManagerService
initiateTaskActionHandler (invoke)
Initiates the TaskActionHandler business process. This process receives the actions from the Worklist application via the TaskManagerService.
receiveUpdatedTask (receive)
Receives the updated task from the TaskActionHandler business process
Compute number of outcomes based on the outcome of the subtask
(switch-case and assign)

Figure 16–35 shows the user task scope modeled using parallel workflow.

Figure 16–35 Parallel Workflow in JDeveloper BPEL Designer



Use Case

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. The process is modeled using the parallel workflow, where each interviewer can vote independently from the other interviewers.

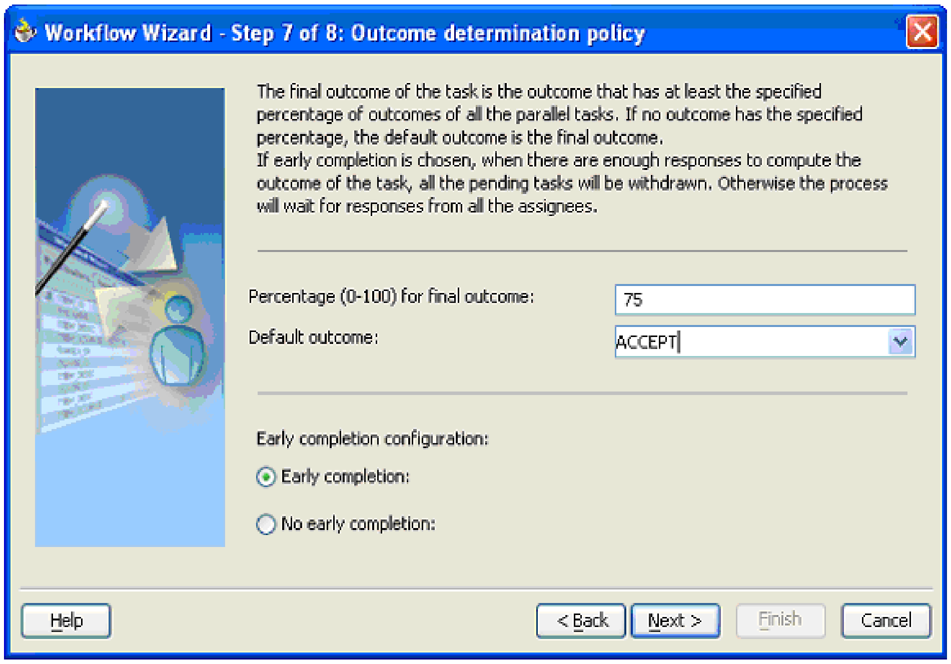
Pattern-Specific Parameters

Parallel workflow has the following pattern-specific parameters.

- The percentage required for an outcome to be chosen as the final outcome
- The default outcome
- Configuration if early completion is enforced

Figure 16–36 shows where these parameters are configured.

Figure 16–36 Outcome Determination



The final outcome of the task is the outcome that has at least the specified percentage of outcomes of all the parallel tasks. If no outcome has the specified percentage, the default outcome is the final outcome.
If early completion is chosen, when there are enough responses to compute the outcome of the task, all the pending tasks will be withdrawn. Otherwise the process will wait for responses from all the assignees.

Percentage (0-100) for final outcome: 75

Default outcome: ACCEPT

Early completion configuration:

[✓ Early completion:]

No early completion:

Customizations for Parallel Workflow

- Changing the outcomes after creation of the user task

  The outcomes are captured in the XML task configuration file, where they can be changed manually. See "Task Outcomes" on page 16-13 for more information.

- Changing the parameters of the outcome determination policy

  The `assign` named `setUserDefinedAttributes` in the generated scope captures the outcome determination policy. The percentage required for the final outcome is assigned to the BPEL variable `oraBPMConclusionPercentage` and the default outcome is set in the BPEL variable `oraBPMDefaultConclusion`.

Parallel Workflow with Final Reviewer

Parallel workflow with a final reviewer is an extension of the parallel workflow in which, after users review the task in parallel, the task is assigned to a final reviewer. The final reviewer can be users or groups. When there is more than one final reviewer or if the task is assigned to a group, one of the users must acquire the task before reviewing it. All that applies to the parallel workflow applies to the parallel workflow with final reviewer as well. The final reviewer can see all the subtasks, where each subtask is a task that was acted on by a user in the parallel workflow.

The final reviewer (users or groups) of the task can be specified in one of the following ways:

1. List of users or groups from OID set during the process design time
2. List of users or groups from another business process variable

The outcome of the final task is selected based on the outcomes of each of the subtasks, as in the parallel workflow. The final reviewer can check the outcome that is selected based on the outcome determination policy from the history of the task from the Worklist Application.

Parallel workflow with final reviewer is implemented using the task continuation concept. See "Task Continuations" on page 16-56 for more information. Parallel workflow with final reviewer is a parallel workflow continued by a simple task.

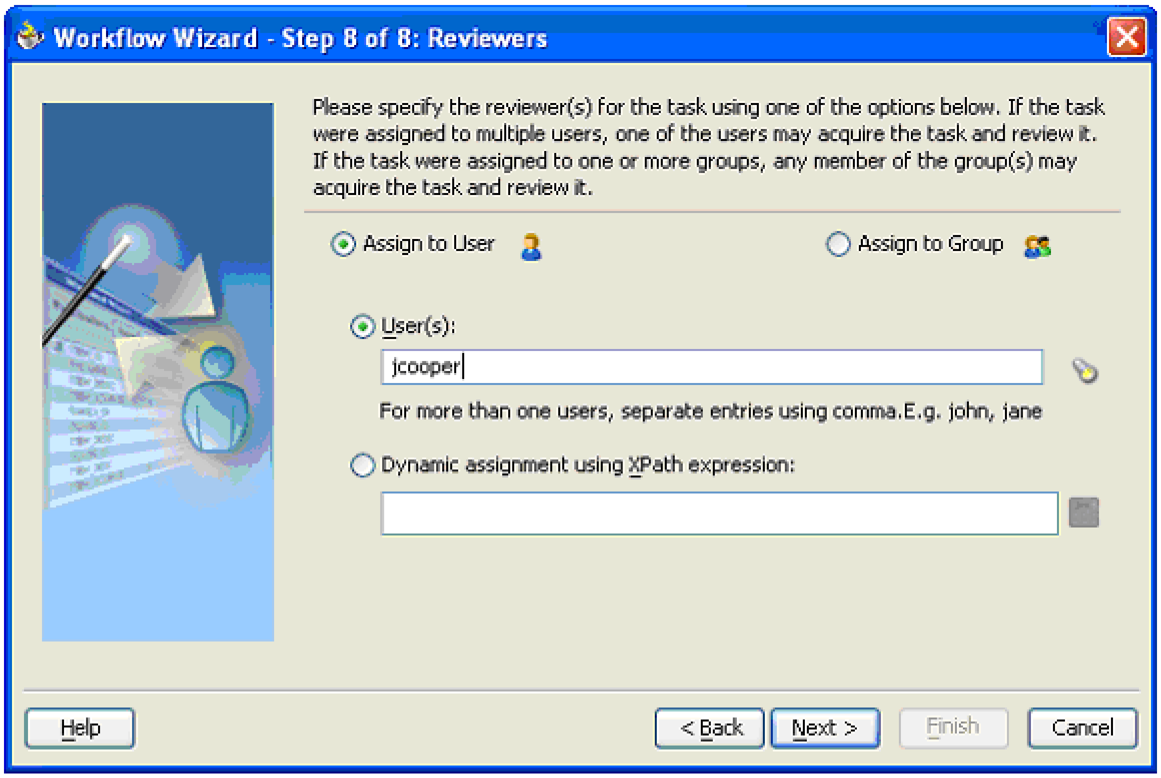
Use Case

A DocumentReview process is used to review a document. The document creator creates the document and initiates a process by specifying the document and the reviewers of the document. Each reviewer can review the document simultaneously and independently of the other reviewers. After all the reviewers are done reviewing, the creator of the document gets to review the comments of the reviewers. See the DocumentReview example in the samples directory for an implementation of this use case.

Pattern-Specific Parameters

Parallel workflow with final reviewer has the same pattern-specific parameters as parallel workflow. In addition to choosing the parallel participants, the final reviewer must also be set. Figure 16–37 shows where the final reviewer is selected.

Figure 16–37 Setting Reviewers



Please specify the reviewer(s) for the task using one of the options below. If the task were assigned to multiple users, one of the users may acquire the task and review it. If the task were assigned to one or more groups, any member of the group(s) may acquire the task and review it.

(●) Assign to User ( ) Assign to Group

(●) User(s): jcooper

For more than one users, separate entries using comma.E.g. john, jane

( ) Dynamic assignment using XPath expression:

Customizations for Parallel Workflow with Final Reviewer

All the customizations for the parallel workflow apply to this pattern, in addition to the following:

- Changing the final reviewer of the task

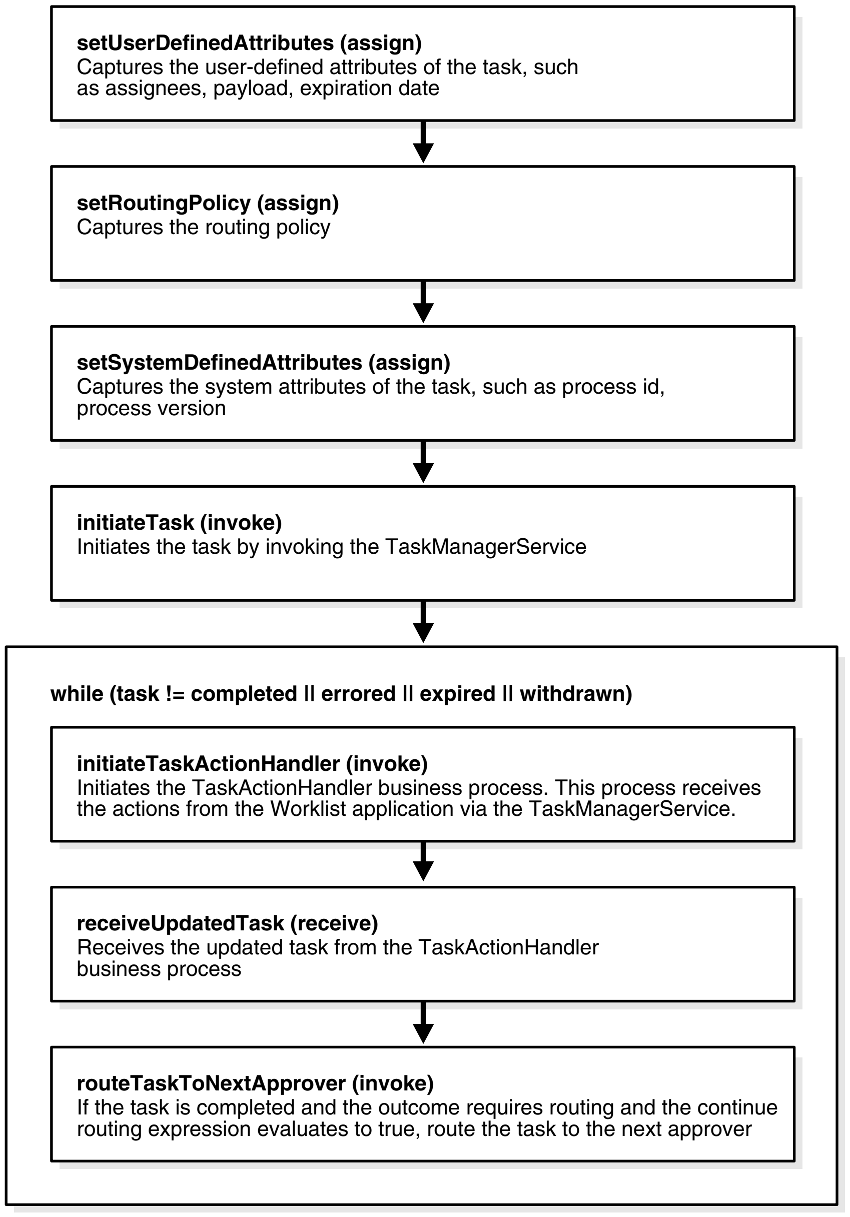
  The `assign` named `setUserDefinedAttributes` in the generated scope captures the reviewers. The reviewers are set in the `/taskmngr:assigneeEntities/taskmngr:assignee` attribute of the `oraBPMReviewers` variable. This assignment can be changed to change the final reviewer.

Adhoc Workflow

Adhoc workflow is used when each user selects users or groups as the next assignee when approving the task. As part of configuring the workflow, only the initial assignee is set. A user can chose either to complete the task by selecting an outcome or to set an outcome and send the task to other users or groups.

Figure 16–38 shows how sequential workflow is implemented in Oracle BPEL Process Manager.

Figure 16–38 Adhoc Workflow


setUserDefinedAttributes (assign)
Captures the user-defined attributes of the task, such as assignees, payload, expiration date
setRoutingPolicy (assign)
Captures the routing policy
setSystemDefinedAttributes (assign)
Captures the system attributes of the task, such as process id, process version
initiateTask (invoke)
Initiates the task by invoking the TaskManagerService
while (task != completed || errored || expired || withdrawn)
initiateTaskActionHandler (invoke)
Initiates the TaskActionHandler business process. This process receives the actions from the Worklist application via the TaskManagerService.
receiveUpdatedTask (receive)
Receives the updated task from the TaskActionHandler business process
routeTaskToNextApprover (invoke)
If the task is completed and the outcome requires routing and the continue routing expression evaluates to true, route the task to the next approver

Use Case

This pattern is typically used when a sequence of users or roles that need to act on the task is not determined automatically by the workflow process, but instead by the user who is currently assigned the task. Each user decides whether the task must be routed further or if it is complete. For example, an HR representative writes a new policy document and has it reviewed by their manager. The manager may decide that another person should also review it before it is accepted. This person may in turn forward the task to others before approving it. Hence, the task may be routed to multiple users before coming back to the original reviewer, who then finally approves it based on comments from others.

Customizations for Adhoc Workflow

- Changing the initial assignee of the task

  The `assign` named `setUserDefinedAttributes` in the generated scope captures the assignment of the task assignee. Depending on whether the task is assigned to a user or a group, the task attribute `assigneeUsers` or `assigneeGroups` is set. This assignment can be changed to modify the initial assignee.

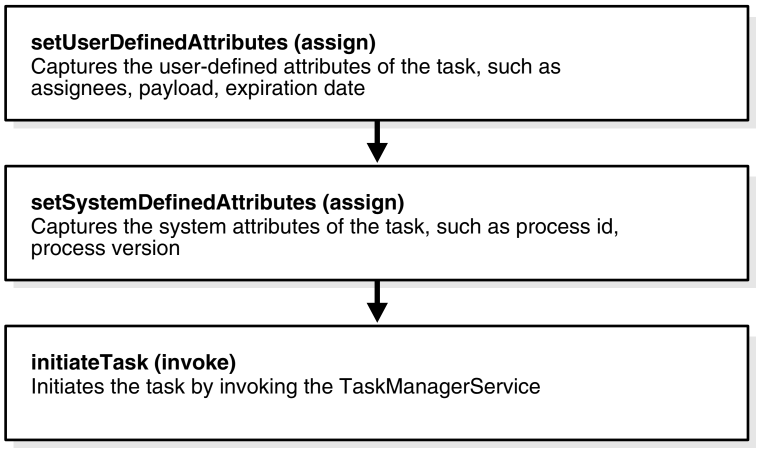
FYI Tasks

The FYI task is a nonblocking task that is listed as an assigned task for the assignee in the Worklist Application. The process that created the task does not wait for a response from the user. The process continues after creating the task. From the Worklist Application, the user can perform the available custom action so that the task no longer appears in the assigned list.

The FYI task cannot be extended. Any other workflow can be extended with an FYI task, but the FYI task itself cannot be extended.

Figure 16–39 shows how the FYI task is implemented in Oracle BPEL Process Manager.

Figure 16–39 The FYI Task


setUserDefinedAttributes (assign)
Captures the user-defined attributes of the task, such as assignees, payload, expiration date
setSystemDefinedAttributes (assign)
Captures the system attributes of the task, such as process id, process version
initiateTask (invoke)
Initiates the task by invoking the TaskManagerService

Use Case

A purchase order approval system processes purchase orders. When the system processes approvals over $100,000, a supervisor must be notified, but without stopping the processing. This information is used by the supervisor to monitor the system.

Customization for FYI Tasks

- Changing the assignee after creating the FYI task

  The `assign` named `setUserDefinedAttributes` in the generated scope captures the assignment of the task assignee. Depending on whether the task was assigned to a user or a group, the task attribute `assigneeUsers` or `assigneeGroups` is set. This assignment can be changed to modify the task assignee.

The User Task 2.0 Macro

The User Task 2.0 Macro supports the user task from earlier releases. This user task requires the business process modeler to assign task properties explicitly and also requires a custom application to view and act on tasks. The User Task 2.0 Macro is available for backward compatibility and is replaced with the workflow services and patterns in this release.

Tasks created using the User Task 2.0 Macro cannot be viewed in the Worklist Application. These tasks cannot be continued with the continue task concept.

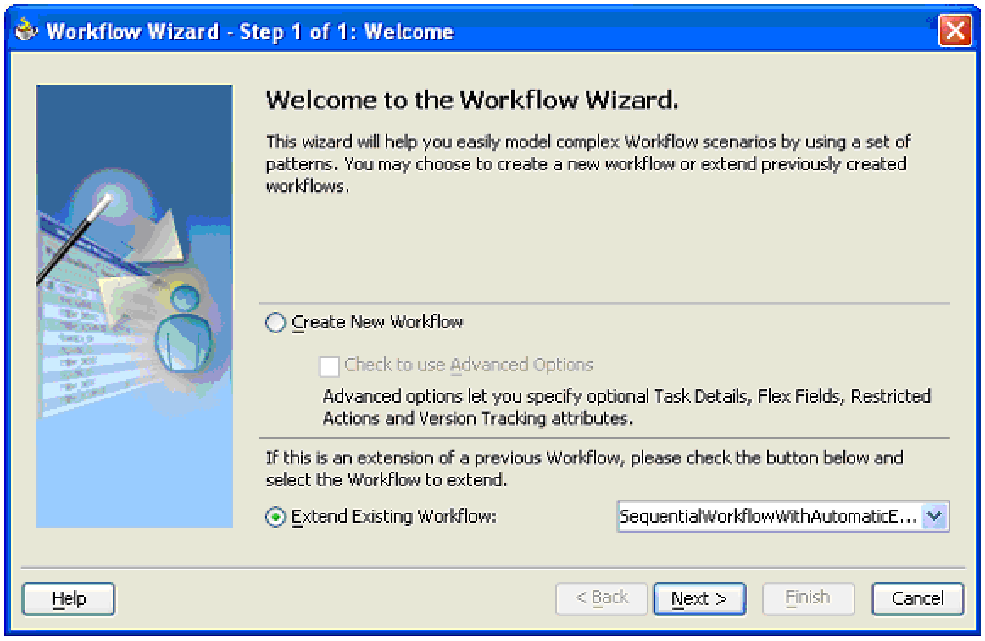
Task Continuations

Complex workflow patterns can be built using the task continuation (extension) concept. Task continuation allows one workflow to be continued with another workflow. When a workflow 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
- Task configuration such as outcomes and notification messages

When a workflow is being created, the first window in the wizard (see Figure 16–40) enables you to determine if a new workflow should be created or a previous workflow should be extended. When **Extend Existing Workflow** is selected, all the existing workflows are listed. Selecting a particular workflow permits the user to extend (continue) the selected workflow.

Figure 16–40 Task Continuation



Welcome to the Workflow Wizard.

This wizard will help you easily model complex Workflow scenarios by using a set of patterns. You may choose to create a new workflow or extend previously created workflows.

○ Create New Workflow

☐ Check to use Advanced Options

Advanced options let you specify optional Task Details, Flex Fields, Restricted Actions and Version Tracking attributes.

If this is an extension of a previous Workflow, please check the button below and select the Workflow to extend.

◉ Extend Existing Workflow: SequentialWorkflowWithAutomaticE...

Any workflow pattern can be extended with any other workflow pattern, with the following restrictions:

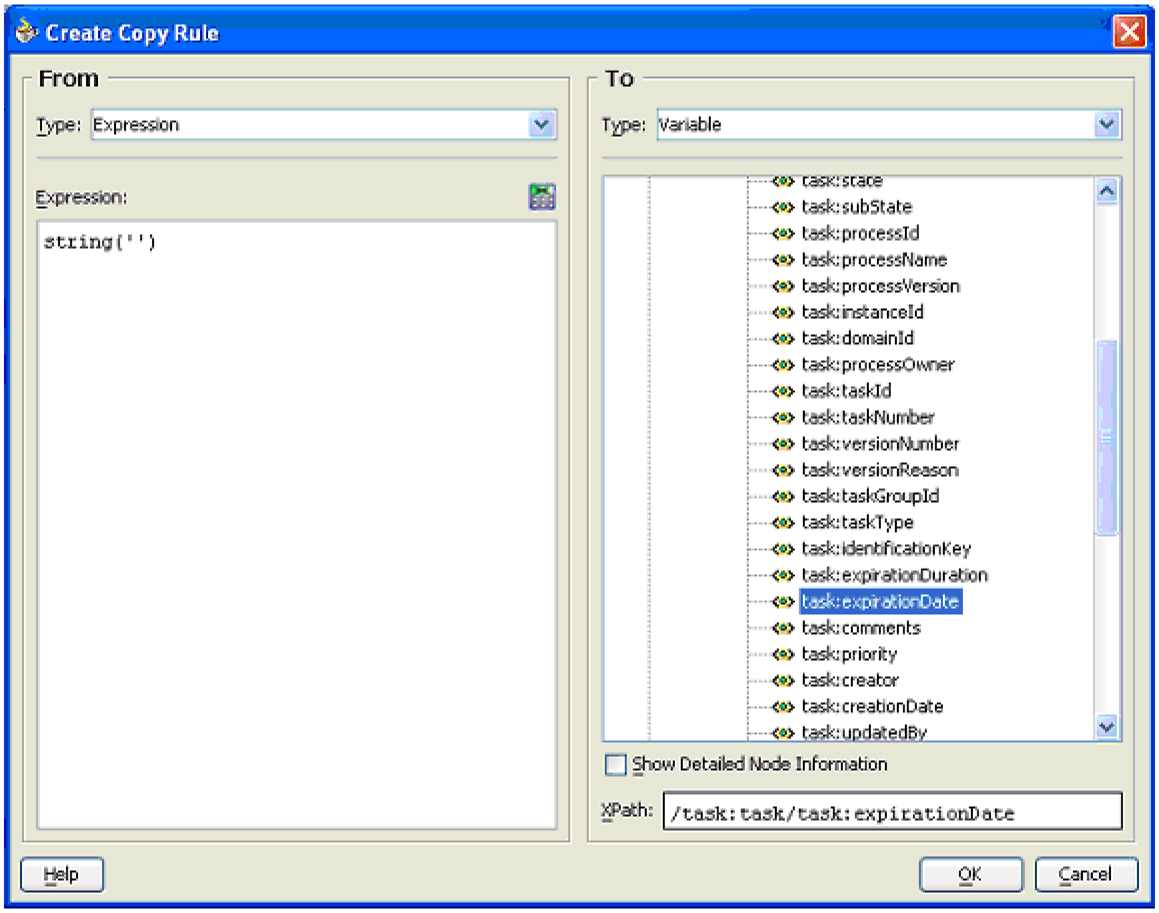
- The FYI task cannot be extended. Any other workflow can be extended with an FYI task, but the FYI task itself cannot be extended.
- The User Task 2.0 Macro cannot be extended.
- No workflow can be extended with the User Task 2.0 Macro.

When a task is extended, the previous task's expiration date is also carried over. This has the following impact:

- If no expiration duration was set in the old task, and
  - if there is no expiration duration set in the extended workflow, then the task does not expire.
  - if a new expiration duration is set in the extended workflow, then this expiration duration is based on the time the extended task is initiated.
- If an expiration duration was set in the old task, and
  - if there is no expiration duration set in the extended workflow, then the expiration date of the old task is the expiration date of the current task.
  - if a new expiration duration is set in the extended workflow, then this expiration duration is based on the previous expiration date (`task:task/task:expirationDate`).

If the expiration date carry over is not required for the scenario being modeled, then the `expirationDate` element must be cleared. You can add a copy in the `setUserDefinedAttributes` of the generated scope to set the `/task:task/task:expirationDate` element to the expression `string('')`. Figure 16–41 shows the copy wizard with such an assignment.

Figure 16–41 Create Copy Rule



```
string('')
```

task:state
task:subState
task:processId
task:processName
task:processVersion
task:instanceId
task:domainId
task:processOwner
task:taskId
task:taskNumber
task:versionNumber
task:versionReason
task:taskGroupId
task:taskType
task:identificationKey
task:expirationDuration
task:expirationDate
task:comments
task:priority
task:creator
task:creationDate
task:updatedBy

XPath: `/task:task/task:expirationDate`

Use Case

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, then 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 parallel workflow for the hiring, and, if the candidate is hired, then a simple workflow can extend the parallel workflow so that the hiring request, history, and the interviewer comments are carried over. This simple workflow is assigned to the HR contact.

Pattern-Specific Parameters

There are no parameters specific to the extended workflow. The parameters are driven by the pattern chosen for the extended workflow.

Customization for Task Continuations

The follow customization applies:

- Customization applicable to the workflow pattern also applies to the extended workflow.
- Changing a new workflow to an extended workflow

Any new workflow can be changed to an extended workflow. For example, if Workflow1 uses the BPEL variable `Workflow1Var` and Workflow2 uses `Workflow2Var`, and Workflow2 must be changed to extend Workflow1, then do the following:

1. Delete the BPEL variable `Workflow2Var`.
2. Replace all occurrences of `Workflow2Var` in the BPEL to `Workflow1Var`.
3. In the generated scope, after the BPEL `assign` activities, there is a scope named `initiateTask`. In this scope, there is an `invoke`, `initiateTask`, that calls the TaskManagerService in the operation `initiateTask`. Change this operation to `reinitiateTask`.

This customization is useful when two workflows that have different payloads must carry the same history information. When modeled as new workflows, the modeler can take advantage of the autogenerated JSP for display purposes and then change the new workflow to an extended workflow.

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` activity is created for each outcome selected during creation of the task. An `otherwise` activity is also generated in the switch to represent cases when the task is withdrawn, expired, or errored.

Payload Updates

The task carries a payload in it. If the payload is set from a business process variable, then an `assign` with the name `copyPayloadFromTask` is created in each of the `case` and `otherwise` activities to copy the payload from the task back to its source. If the payload is expressed as other XPath expressions (such as `ora:mergeChildNodes(...), ora:getNodes(...)`), then this `assign` is not created because of the lack of a process variable to copy the payload back. If the payload need not 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` activity. 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.

```
<switch name="taskSwitch">
  <case condition="bpws:getVariableData('SequentialWorkflowVar1',
'/task:task/task:state') = 'COMPLETED' and
bpws:getVariableData('SequentialWorkflowVar1', '/task:task/task:conclusion') =
'ACCEPT'">
```

```
    <bpelx:annotation>
      <bpelx:pattern>Task outcome is ACCEPT
      </bpelx:pattern>
    </bpelx:annotation>
      ...
  </case>
  <case condition="bpws:getVariableData('SequentialWorkflowVar1',
'/task:task/task:state') = 'WITHDRAWN'">
    <bpelx:annotation>
      <bpelx:pattern>Task is withdrawn
      </bpelx:pattern>
    </bpelx:annotation>
     ...
  </case>
  <case condition="bpws:getVariableData('SequentialWorkflowVar1',
'/task:task/task:state') = 'EXPIRED'">
    <bpelx:annotation>
      <bpelx:pattern>Task is expired
      </bpelx:pattern>
    </bpelx:annotation>
     ...
  </case>
  <case condition="bpws:getVariableData('SequentialWorkflowVar1',
'/task:task/task:state') = 'ERRORED'">
    <bpelx:annotation>
      <bpelx:pattern>Task is errored
      </bpelx:pattern>
    </bpelx:annotation>
     ...
  </case>
  <otherwise>
    <bpelx:annotation>
      <bpelx:pattern>Task is EXPIRED, WITHDRAWN or ERRORED
      </bpelx:pattern>
    </bpelx:annotation>
      ...
  </otherwise>
</switch>
```

Task Notifications

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, or SMS.

The notifications for a task can be configured during the creation of a task. 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 actions:
  - Task is assigned to a user
  - Task is assigned to a new user in a sequential workflow
  - Task is renewed
  - Task is delegated
  - Task is reassigned

  - Task is escalated
  - Task is resumed (from a suspension)
  - Information for a task is submitted
- Information is requested for a task
- Task is suspended
- Task is errored
- Task is expired
- Task is completed
- Task is withdrawn
- Reminder

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
- Creator—the user who created the task
- Approvers—the users who have approved the task so far
  - This applies in a sequential workflow where multiple users have approved the task and a notification must be sent to all of them.

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.

See Also: Chapter 15, "Oracle BPEL Process Manager Notification Service"

Channels Used for Notification

The channel through which a user is notified is determined by the notification preference attribute of the user as 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 `users-properties.xml` at

```
Oracle_Home\integration\orabpel\system\services\config
```

In an OID-based system, the user properties can be changed using Oracle Delegated Administration Service. If this attribute is not set, the e-mail channel is used as the default.

Notification Messages

Notification messages can include static strings and XPath expressions as follows:

```
<html>
<body>
<%bpws:getVariableData('TaskNotification',
'/taskNotification:taskNotification/taskNotification:recipientDisplayName')%>
<br>
<%bpws:getVariableData('taskObject', '/task:task/task:title')%> is assigned to
you. Please act on the task from the <A
href="<%bpws:getVariableData('TaskNotification',
'/taskNotification:taskNotification/taskNotification:worklistApplicationLink')%>”>
```

```
Worklist Application</A>
</body>
</html>
```

In creating the messages, only two BPEL variables are available to the user—the task variable and a task notification variable. 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 the XPath tree browsing. The task notification variable is a transient variable available to get information about the recipient and assignees.

Table 16–1 lists the elements that are available when using the task notification variable.

Table 16–1 Elements for the Task Notification Variable

Element Name	Description
`recipient`	The recipient id
`recipientLocale`	The locale of the recipient to use in getting messages from resource bundles for internationalization. Messages from bundles can be retrieved using the `orcl:get-localized-string(…)` extension function.
`recipientDisplayName`	The display name of the recipient as in the user store (OID, Active directory, and so on). If no display name is specified, the convention *`{lastname}, {firstname}`* is used.
`assignees`	A comma-separated string of all the assignee ids.
`assigneeDisplayNames`	A comma-separated string of all the assignee display names.
`worklistApplicationLink`	The link to the Worklist Application. This link takes the user to the particular task for which the e-mail is sent.
`actionLinks`	If actionable notifications are enabled, the actions link has text to create HTML links of the format `<html link>custom action<html link><space><html link>custom action<html link><space>`….
`processURL`	The process URL is useful to look up a resource bundle that is deployed with the BPEL suitcase. Messages from resource bundles can be retrieved using the `orcl:get-localized-string(…)` extension function. The first argument in this function is the base URL. If the message bundle is deployed with the suitcase, the base URL can be retrieved using the `processURL` element of the task notification BPEL variable.

Any XPath extension function that can only be evaluated within the context of a business process such as `ora:getProcessURL()` or `ora:getInstanceId()` cannot be evaluated in the task notifications. Task attributes such as `processName`, `processVersion`, `instanceId`, and `domainId` can be used for these purposes.

E-mail Approval

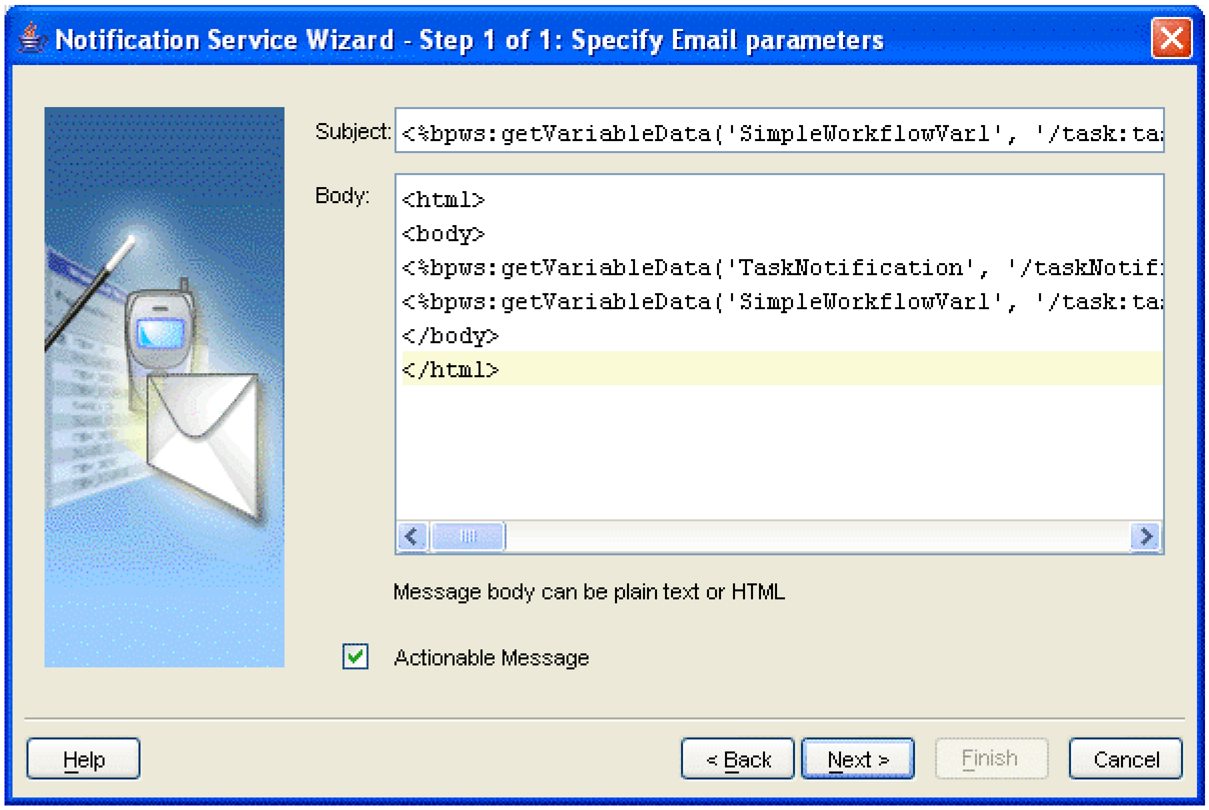
Task actions can be performed through e-mail, if the task is set up to enable e-mail actions. (The same actions can also be performed from the Worklist Application.) An actionable e-mail account is the account in which task action-related e-mails are received and processed. This e-mail account name is identified by the element `actionableEmailAccount` in the XML file *`Oracle_Home`*`\integration\orabpel\system\services\config\wf_config.xml`.

For example:

```
<workflowConfigurations
   xmlns="http://xmlns.oracle.com/pcbpel/humanworkflow/configurations">

   <actionableEmailAccount>AccountReceiving</actionableEmailAccount>
    ...
</workflowConfigurations>
```

Figure 16–42 shows where you specify if the e-mail message is an actionable message.

Figure 16–42 Notification Service Wizard - Step 1 of 1: Specify Email Parameters


Notification Service Wizard - Step 1 of 1: Specify Email parameters
Help
< Back
Next >
Finish
Cancel

Subject: <%bpws:getVariableData('SimpleWorkflowVar1', '/task:ta

Body:

```
<html>
<body>
<%bpws:getVariableData('TaskNotification', '/taskNotif
<%bpws:getVariableData('SimpleWorkflowVar1', '/task:ta
</body>
</html>
```

Message body can be plain text or HTML

✓ Actionable Message

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 marked for reminders (REMINDER) is used, and if there is no such message, the message meant for the assignees when the task is assigned is used to send reminders.

Reminders can be added in the task configuration file. In the task configuration file, the reminder element is added inside the notifications element as follows:

```
<taskType ......... >

    <task actionableNotifications="YES">
    ..........
    </task>

    <notifications>
        <reminder recurrence="1" relativeDate="ASSIGNED">PT3H</reminder>
        <action name="REMINDER">
            <messages recipient="ASSIGNEES"

xmlns:ns="http://xmlns.oracle.com/ias/pcbpel/NotificationService">

            ............

            </messages>
        </action>
        <action name="ASSIGNED">
            <messages recipient="ASSIGNEES"

xmlns:ns="http://xmlns.oracle.com/ias/pcbpel/NotificationService">

            ............

            </messages>
        </action>
    </notifications>
</taskType>
```

The `recurrence` specifies the number of times reminders are sent. The possible values for recurrence are `EVERY`, `NEVER`, `0`, `1`, `2` ..., `10`.

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`.

The value for the reminder element itself is 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`, `PT130S`, `PT2M10S`, `P1DT2S`, `-P1Y`, or `P1Y2M3DT5H20M30.123S`.

The following examples illustrate when reminders are sent.

- 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, 3/27/2005 10:00 AM, and so on until the user acts on the task.
- If the `relativeDate` is `EXPIRED`, 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 `EXPIRED`, 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.

Payload Display

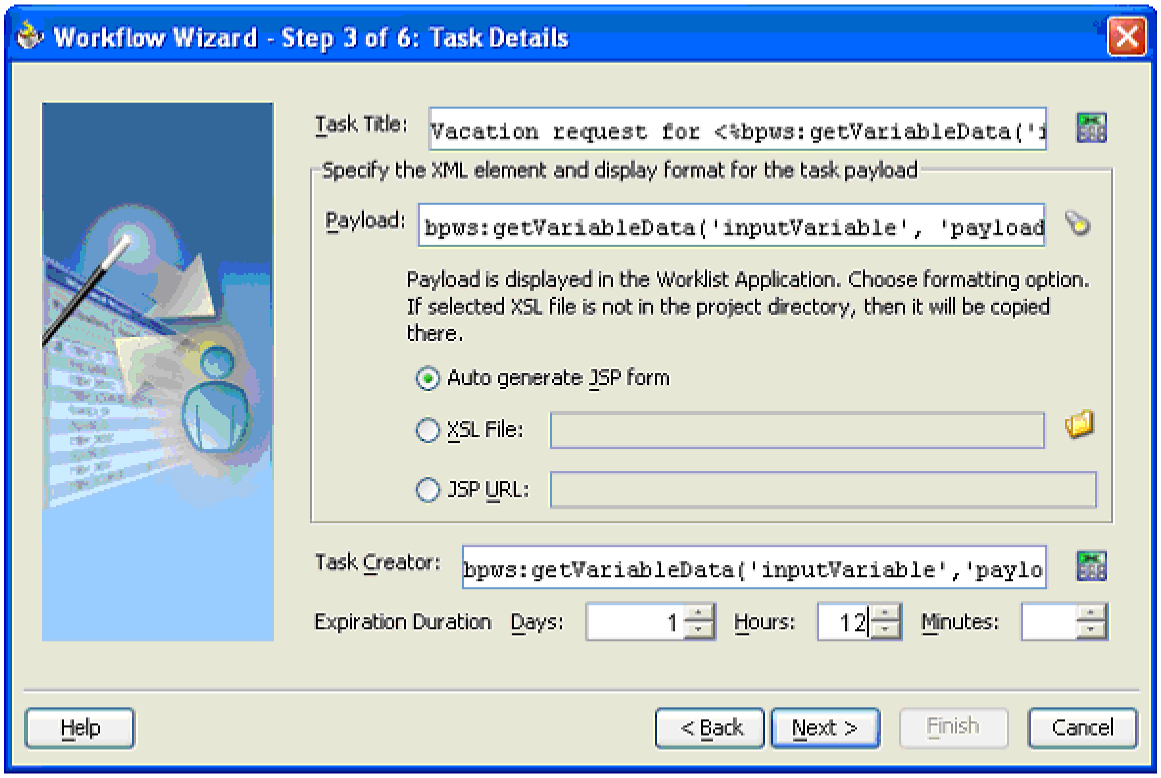
The task payload is an XML structure that must be displayed in a usable form in the Worklist Application. You have three options for displaying the payload in the Worklist Application:

- Autogenerated JSP
- XSL

- The Custom JSP URL

As Figure 16–43 shows, any of these options can be selected as the payload display mechanism in the Workflow wizard. The **Payload** field is expecting a `bpws:getVariableData` function that gets the payload root from a BPEL process's variable data.

Figure 16–43 Workflow Wizard - Step 3 of 6: Task Details



Task Title: Vacation request for <%bpws:getVariableData('i

Specify the XML element and display format for the task payload

Payload: bpws:getVariableData('inputVariable', 'payload

Payload is displayed in the Worklist Application. Choose formatting option. If selected XSL file is not in the project directory, then it will be copied there.

[✓ Auto generate JSP form]

XSL File:

JSP URL:

Task Creator: bpws:getVariableData('inputVariable','paylo

Expiration Duration Days: 1 Hours: 12 Minutes:

Autogenerated JSP

When the **Auto generate JSP form** option is selected, two files, a default JSP file and a mapping file, are automatically generated to display the payload at the end of the Workflow wizard.

The name of the default JSP file, *`task_name`*`_WF_Form.jsp`, is generated based on your task name. The file is added to the HTML root directory of your project, which is by default the `public_html` directory.

Along with the default JSP file, a mapping XML file is also generated. It is named *`task_name`*`_WF_Fields.xml`, and is added to the root directory of your project.

Behind the scenes, the JSP run-time library and the OraBPEL library are automatically added to your BPEL project for compilation of the JSP file.

The default JSP shows all the simple types in the payload XSD. It does not show attributes. 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 that are copied from variables that are not simple types are supported. This is to say, in the payload field in the preceding picture, 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.

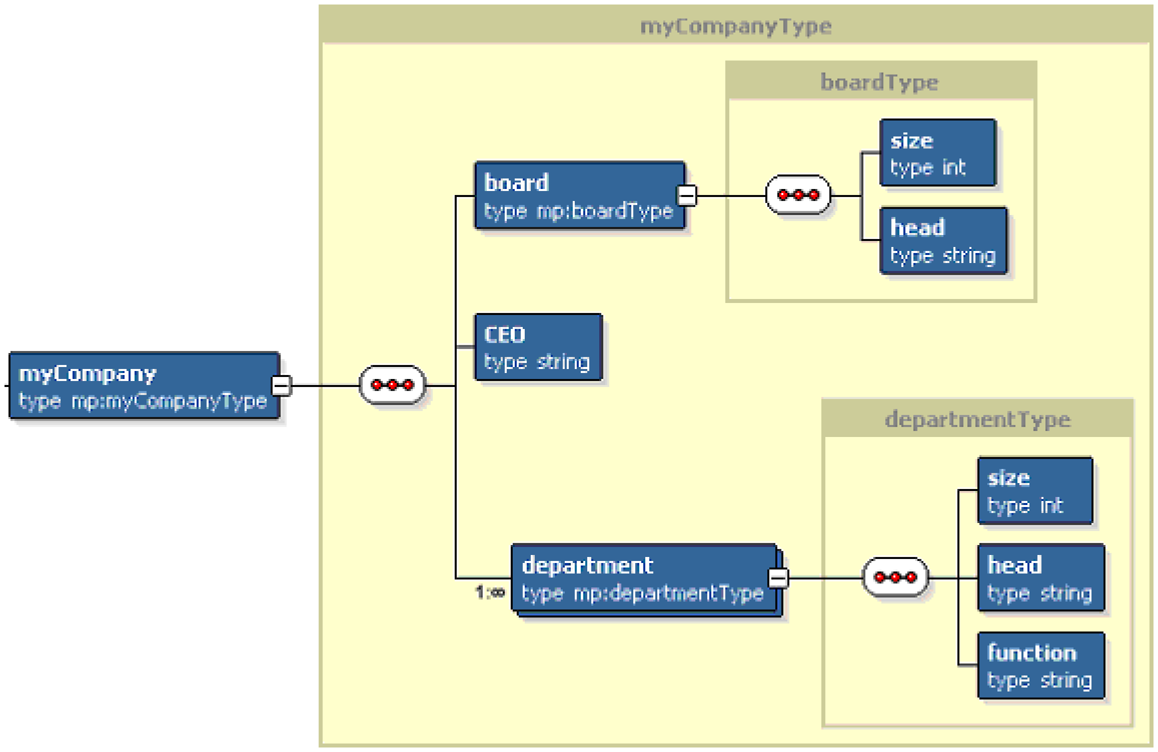
Default JSP groups the whole payload structure. It groups simple types that belong to the same parents.

For example, assume the user provides the following payload XSD:

```
<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>
```

This XSD has the structure shown in Figure 16–44.

Figure 16–44 Structure of the XSD for myCompanyType



In the default JSP, based on the structure of the leaf nodes, there are three sections: {size, head}, {CEO}, and {size, head, function}. These three sections are named according to their parents' names; that is, the sections are named `board`, `my Company`, and `department`, respectively. Because the section department can have multiple occurrences, all the fields in this section, that is, size, head, and function, are horizontally presented in a table, with each row representing one department.

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 showable fields. The root element in the mapping file contains its `targetNameSpace`, other namespaces, `showXmlView`, and `xmlEditable` as its attributes.

The payload, by default, shows only the HTML form view. It also has an XML source view that can be used to set payload XML directly. `ShowXmlView` identifies if the XML source view is enabled, while `xmlEditable` identifies if the XML source view in the payload presentation is editable or not. `ShowXmlView` is set to `false` by default, while `xmlEditable` is set to `true`.

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. To do that, you modify your `taskConfig`*`taskName`*`.xml` file. Add two attributes—`resourceBundleName` and `resourceBundleLocation`—to the top element, `task`. The `resourceBundleLocation` attribute points to a JAR file, and `resourceBundleName` specifies the resource bundle's file name in the JAR file. The `resourceBundleLocation` attribute is optional for globalization purposes.

The following code shows an example of `taskConfig<`*`taskName`*`>.xml`:

```
<taskType name="taskConfigSimpleWorkflow2.xml"
     resourceBundleName="MyBundle"
     targetNamespace="http://taskTypes/taskConfigSimpleWorkflow2.xml"
     features="xpathNotification"
     xmlns:tns="http://taskTypes/taskConfigSimpleWorkflow2.xml"
     xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
     xmlns:xp20="http:
//www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.Xpath20"
     xmlns:ldap="http://schemas.oracle.com/xpath/extension/ldap"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
     xmlns:Notification="http:
//xmlns.oracle.com/pcbpel/taskservice/taskNotification"
     xmlns:client="http://xmlns.oracle.com/i18nProcess"
     xmlns:ora="http://schemas.oracle.com/xpath/extension"
     xmlns="http://xmlns.oracle.com/pcbpel/taskservice/tasktype"
     xmlns:ns="http://xmlns.oracle.com/ias/pcbpel/NotificationService"
     xmlns:bpelx="http://schemas.oracle.com/bpel/extension"
     xmlns:task="http://xmlns.oracle.com/pcbpel/taskservice/task"
     xmlns:orcl="http:
//www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.ExtFunc">
  <task actionableNotifications="YES"
     xmlns="http://xmlns.oracle.com/pcbpel/taskservice/tasktype">
    <conclusions>
      <conclusion name="ACCEPT">
        <displayValue>getMessage(ACCEPT_MSG)</displayValue>
      </conclusion>
...
</taskType>
```

`MyBundle` 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 = Accept0
REJECT_MSG = Reject0
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

```
<field>
   <xpath>/ns0:task/ns0:payload/taskTitle</xpath>
   <editable>true</editable>
   <resourceKey>TASK_TITLE</resourceKey>
</field>
```

then calling

```
PayloadUtil.getElementDisplayName("/ns0:task/ns0:payload/taskTitle", ,form,
context.getLocale(), task)
```

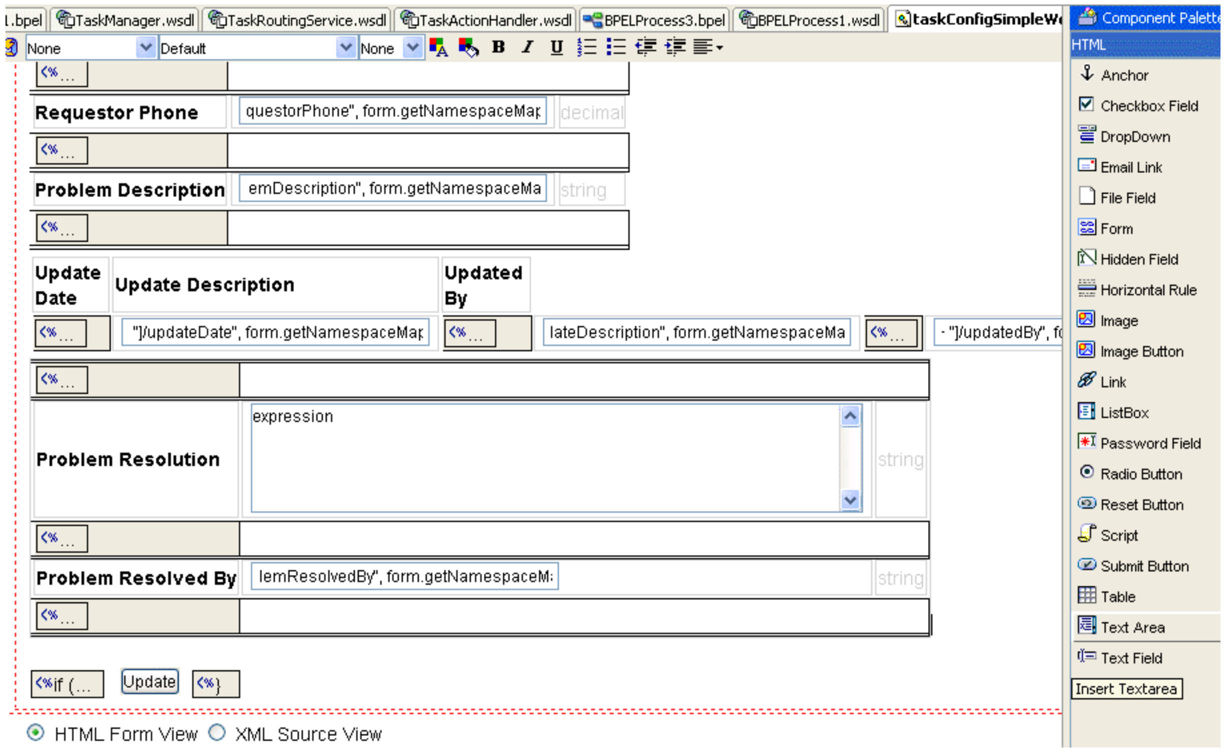
in the default JSP returns the string `i18n Task` if your locale is set to `en-US`. Similarly, if your locale is set to French, the proper properties file (`MyBundle_fr.properties`) is picked up.

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 JDeveloper BPEL Designer.

By default, all the fields are set to text field. If you want to change text field to text area, you can do the following. In the **Component Palette**, select **Text Area**, as shown in Figure 16–45. Drop it into the position of the text field you want to replace. Note that the name of the text field is set by calling the function `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. Set the text area's name and value to the same name and value as text field. Delete the text field.

Figure 16–45 JDeveloper BPEL Designer JSP Design View



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**.

If the place you want to insert HTML elements that are 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.

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. 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 `UpdateServlet` 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. See "The Custom JSP URL" on page 16-72 for a lightweight code analysis of the JSP.

> **See Also:** The `HelpDeskServiceRequest` demo in `Oracle_Home\integration\orabpel\samples\demos` for an example of an autogenerated JSP and how to change the payload presentation

By putting the statement `<%@ page pageEncoding="UTF-8" %>` in the default JSP, UTF-8 is set as the default encoding. If you want to modify this statement to set the JSP to other encoding, for example ISO, then you must also change `UpdateServlet`. To do this, add one encoding element in the `taskConfigtask_name.xml` file. The modified file looks as follows:

```
<taskType>
   <task>
      ...
      <payloadDisplay>
         <jsp>defaultWFPayloadJSP</jsp>
         <xpathFile>taskConfigSimpleWorkflowWithAutomaticEscalation1_WF_
Fields.xml</xpathFile>
         <encoding>ISO</encoding>
      </payloadDisplay>
    </task>
</taskType>
```

Multibyte Payload in the Task Detail JSP

To ensure that a multibyte payload is correctly displayed on the Task Detail page, modify the `TaskType.xml` file that is generated during design time by adding the following element as a child element of `PayloadDisplay`:

```
<encoding>UTF-8</encoding>
```

Deploying the Autogenerated JSP

Using JDeveloper BPEL Designer, you can deploy your BPEL project directly to OC4J. When you deploy your project, the default JSP file is deployed at

`Oracle_Home``\orabpel\system\appserver\oc4j\j2ee\home\applications\hw_services\worklistxpress\payload\`*`bpel_process_name_bpel_process_version`*`\`

Because every autogenerated JSP is named based on its task name only, by deploying the JSPs under their corresponding *`bpel_process_name_bpel_process_version`* directories, there are no collisions.

On other platforms, the same directory structure is used.

If you choose to deploy your project on OC4J by using `obant`, ensure that you copy the autogenerated JSP file to the preceding directory by inserting a `copy` statement like the following to your project's `build.xml` file.

```
        <copy todir="${home}/system/appserver/oc4j/j2ee/home/applications/hw_
services/worklistxpress/payload/bpel_HelpDeskServiceRequest_1.0">
            <fileset dir="public_html">
                <include name="*.jsp" />
            </fileset>
        </copy>
```

If you make changes to your JSP or mapping file, you must deploy your BPEL project after the modifications.

> **See Also:** The `HelpDeskServiceRequest` demo in *`Oracle_Home`*`\integration\orabpel\samples\demos` for more information on the `build.xml` file

XSL

Another payload display option is to provide an XSL file that transforms the payload XML instance to HTML. The root of the XML instance that is to be transformed is the `{http://xmlns.oracle.com/pcbpel/taskservice/task}payload` element. The payload itself cannot be updated using this option.

See Also: `Oracle_Home\integration\orabpel\samples\demos\OrderApproval` for an example in which an XSL file is used to display the payload

The Custom JSP URL

A custom JSP can display the payload in the Worklist Application. This section describes how to write a JSP for payload presentation. Before deciding on writing a custom JSP, evaluate if the default payload JSP can be modified to suit your needs, since modifying the default JSP is the easiest way to construct your JSP. See "Customizing the Autogenerated JSP" on page 16-67 for how to generate and modify the default payload JSP.

Code analyzing an autogenerated payload JSP helps you formalize how to write your JSP. The autogenerated JSP imports `oracle.tip.pc.api.worklist.payload.*`. This package contains classes used to access payload values.

The first section of the default JSP gets the payload object from the task object. No null checking is done for these variables in the JSP file because these checks are done before calling the JSP.

A form object is created from the task object. This form object represents the mapping file. The optional login page, next page, and error page are received from the parameter list in case they are needed in the JSP.

The call to `PayloadFormGenerator.getRequiredFormParameters()` returns a map of required parameters that must be sent to the JSP and the update payload servlet. This map contains parameter names (`keys`) and parameter values (`values`). Ensure that, in the payload JSP, you call this method and send these parameters so that the update payload servlet functions properly.

The next section contains code that presents the payload. The first `div` block gives the XML view of the payload, while the second `div` block provides the HTML form view. Notice that the name of every form parameter is the result of calling `PayloadFormGenerator.constructName(xpath)`. This is because there are illegal characters in XPath while used as HTTP parameters. Ensure that the JSP calls this function to construct parameter names for the particular XPath.

`Form.getField(xpath)` gets the field from the mapping file. Since the custom JSP does not use a mapping file, this function need not be called.

`PayloadFormGenerator.selectNodeValue()` is the function that gets the value of the given XPath from the payload. The namespace map must be passed to this function.

Your JSP can use the update servlet provided in the application.

`PayloadConstant.UPDATE_SERVLET_URL` stores the URL of the payload update servlet.

The update servlet expects the following required parameters to be set:

```
PayloadConstant.WORKLIST_CONTEXT_PARAMETER_NAME
PayloadConstant.WORKLIST_TASKID_PARAMETER_NAME
```

And these optional parameters:

```
PayloadConstant.WORKLIST_TASK_VERSION_PARAMETER_NAME (optional)
PayloadConstant.WORKLIST_ERROR_PAGE_PARAMETER_NAME (optional)
PayloadConstant.WORKLIST_LOGIN_PAGE_PARAMETER_NAME (optional)
PayloadConstant.WORKLIST_NEXT_PAGE_PARAMETER_NAME (optional)
```

The preceding parameter names and their corresponding values can be obtained by calling the `PayloadFormGenerator.getRequiredFormParameters()` function.

It also expects certain parameters to be defined in the servlet's session object, but you need not be concerned about these parameters in your JSP file.

For all payload updates, the custom JSP must send the names and values of the fields that have to be updated. The names are constructed by calling `PayloadFormGenerator.constructName(String xpath).`

The update servlet must know what the prefixes represent in the XPath. Therefore, a namespace map must be sent to the servlet. Any parameter whose name starts with `ns` and is followed by a numeric number is assumed to be a namespace prefix. Its value is the namespace value. You should always send parameter `ns0` with its value `PayloadConstant.NS_ROOT_URL`.

APIs

As with the autogenerated JSP, the JSP run-time library and the OraBPEL library are automatically added to your BPEL project for compilation of the custom JSP file.

> **See:** *`Oracle_ Home`*`\integration\orabpel\docs\workflow\oracle\tip\pc \api\worklist\payload` for Javadoc API documentation

Customizing the Complete Task JSP

The previous sections described how you can customize the look and feel of the payload portion of the task using a custom JSP, XSL, or by modifying the autogenerated JSP. You can also override the complete JSP, including the header (task attributes and actions section) and the footer (attachment, comments section) by specifying a custom JSP to display the entire task details. The payload JSP URL is captured in the task configuration XML file in the element `payloadDisplay`, as follows:

```
<taskType ...>
  <task ...>
   ...
    <payloadDisplay>
      <jsp>http://localhost:9700/payloads/vacationrequest/payload.jsp</jsp>
    </payloadDisplay>
      ...
  </notifications>
</taskType>
```

Configuration for Task Service

The following sections discuss how to configure task services.

Autorelease Duration

If a task is assigned to groups or multiple users, one of the users in the group or the list of users must acquire the task before acting on it. After the task is acquired, none of the initial assignees can see the task if the task was assigned to them. If the user does not act within a given time, the task is automatically released so that all the other users in the group or list of users can see it. A particular business process can disable the autorelease by making autorelease a restricted action. See "Restricted Actions" on page 16-16 for more information.

The release duration is configurable in the file `wf_config.xml` at

```
Oracle_Home\integration\orabpel\system\services\config
```

The configurations for the autorelease durations are in the element `taskAutoReleaseConfiguration`. The release durations can be configured for tasks of each priority. For each priority, the autorelease duration can be specified as a percentage of the expiration (the `percentageOfExpiration` attribute) duration or a default value (the `default` attribute). The default values are used when the task does not have an expiration duration. The datatype of `default` 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`, `PT130S`, `PT2M10S`, `P1DT2S`, `-P1Y`, or `P1Y2M3DT5H20M30.123S`.

For example, if the task of priority `3` is acquired at 3/24/2005 10:00 AM and the task expires at 3/31/2005 10:00 AM, then the time left for expiration is 7 days. If the `percentageOfExpiration` for priority 3 tasks were 50, then the task is released at 3/37/2005 10:00 PM (3 ½ days from when it was acquired).

The `taskAutoReleaseConfiguration` element in the configuration file is shown as follows:

```
<workflowConfigurations
   xmlns="http://xmlns.oracle.com/pcbpel/humanworkflow/configurations">

   <taskAutoReleaseConfigurations>
      <taskAutoRelease priority="1" default="P1D" percentageOfExpiration="30"/>
      <taskAutoRelease priority="2" default="P2D" percentageOfExpiration="40"/>
      <taskAutoRelease priority="3" default="P3D" percentageOfExpiration="50"/>
      <taskAutoRelease priority="4" default="P4D" percentageOfExpiration="60"/>
      <taskAutoRelease priority="5" default="P5D" percentageOfExpiration="70"/>
   </taskAutoReleaseConfigurations>

</workflowConfigurations>
```

Actionable E-mail Accounts

Task actions can be performed through e-mail. The 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 property `oracle.tip.pc.services.hw.taskservice.actionableEmailAccount` at

```
Oracle_Home\integration\orabpel\system\services\config\pc.properties
```

Worklist Application URL

In the e-mails that are sent for tasks, the link to the Worklist Application is read from the XML file `wf_config.xml` at

```
Oracle_Home\integration\orabpel\system\services\config\
```

The element `worklistApplicationURL` identifies the URL. Configuring this is useful if the custom Worklist Application is built. The tag `PC_HW_TASK_ID_TAG` in this URL is replaced with the task id when constructing the URL for the e-mail.

```
<workflowConfigurations
   xmlns="http://xmlns.oracle.com/pcbpel/humanworkflow/configurations">

   <worklistApplicationURL >
http://localhost:9700/integration/worklistapp/TaskDetails?taskId=PC_HW_TASK_ID_TAG
```

```
</worklistApplicationURL >
...
</workflowConfigurations>
```

Identity Service

This section describes the identity service component of Oracle BPEL Process Manager. Identity service is a thin Web service layer on top of the Oracle Application Server 10*g* security infrastructure, namely OracleAS JAAS Provider (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

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 need not 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.

  > **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, then 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 process owner can change the outcome of the task.
- BPMSystemAdmin—Both BPMWorkflowAdmin and BPMSystemAdmin have the same level of workflow privileges.

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.

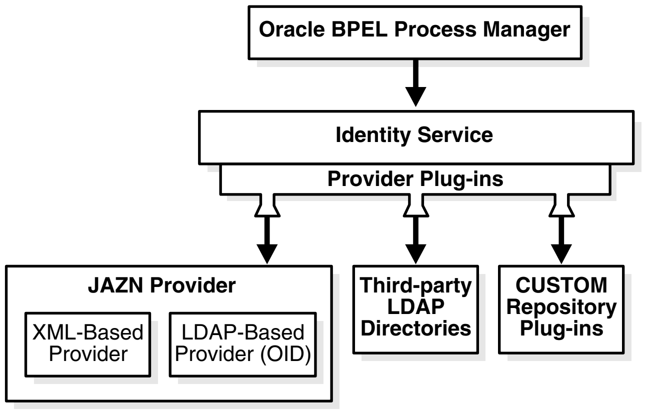
The following table represents the relationship between the grantees and roles:

Role\Grantee	bpeladmin	default	guest	BPMWorkflowAdmin	BPMSystemAdmin
BPMSystemAdmin	Directly	--	--	--	--
BPMWorkflowAdmin	Indirectly through BPMSystemAdmin	--	--	--	Directly
BPMWorkflowReassign	Indirectly through BPMSystemAdmin	--	--	Directly	Indirectly through BPMWorkflowAdmin
BPMWorkflowSuspend	Indirectly through BPMSystemAdmin	--	--	Directly	Indirectly through BPMWorkflowAdmin
BPMWorkflowViewHistory	Indirectly through BPMSystemAdmin	--	--	Directly	Indirectly through BPMWorkflowAdmin

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 16–46.

Figure 16–46 Identity Service Providers


Oracle BPEL Process Manager
Identity Service
Provider Plug-ins
JAZN Provider
XML-Based Provider
LDAP-Based Provider (OID)
Third-party LDAP Directories
CUSTOM Repository Plug-ins

The identity service providers are used to 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 inquires to the JAZN layer. Two JAAS providers are supplied as part of the OC4J security infrastructure: the XML-based file and LDAP-based OID.

See Also: *Oracle Application Server Containers for J2EE User's Guide*

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 are used to retrieve these details.

Custom User Repository Plug-ins

This mode enables UPI to define a new custom identity service provider to plug in a specific non-LDAP-based user repository. 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 "User and Role Properties" on page 16-78 for more information.

> **See Also:** See `*Oracle_Home*\integration\orabpel\docs\workflow\oracle\tip\pc\services\identity` for Javadoc on the `BPMIdentityService` interface

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 users and groups 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 realmName 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 the XML-based provider, then you must supply a username and password to the Admintool; for details see Authentication and the JAZN Admintool (XML-based provider only)". If you are using the LDAP-based provider, you need not specify the `-user` and `-password` arguments.

- If you are using a third-party LDAP server or a custom user repository, you must use the specific tools available for that directory.

User and Role Properties

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
- Time zone
- Preferred language
- 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. Also, automatic escalation and manager views are not available if the manager field is not available to identity service.

The following OID `objectClasses` are used to specify user and role properties such as mail, manager, and telephoneNumber.

- top
- person
  - cn
  - sn
  - description
  - telephoneNumber
- organizationalPerson
  - title
  - telephoneNumber
  - facsimileTelephoneNumber
- inetOrgPerson

- displayName
- givenName
- manager
- mail
- homePhone
- mobile
- pager
- preferredLanguage

- orclUserV2
  - middleName
  - orclTimeZone
  - orclWorkflowNotificationPref
- orclGroup

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

Oracle_Home`\integration\orabpel\system\services\config`

Configuring Identity Service

The following sections describe how to configure identity services.

Structure of the Identity Service Configuration File

Identity service configuration is defined in the `is_config.xml` file. The file must be located in a directory that is included in the `CLASSPATH` of Oracle BPEL Process Manager. By default, it is stored in

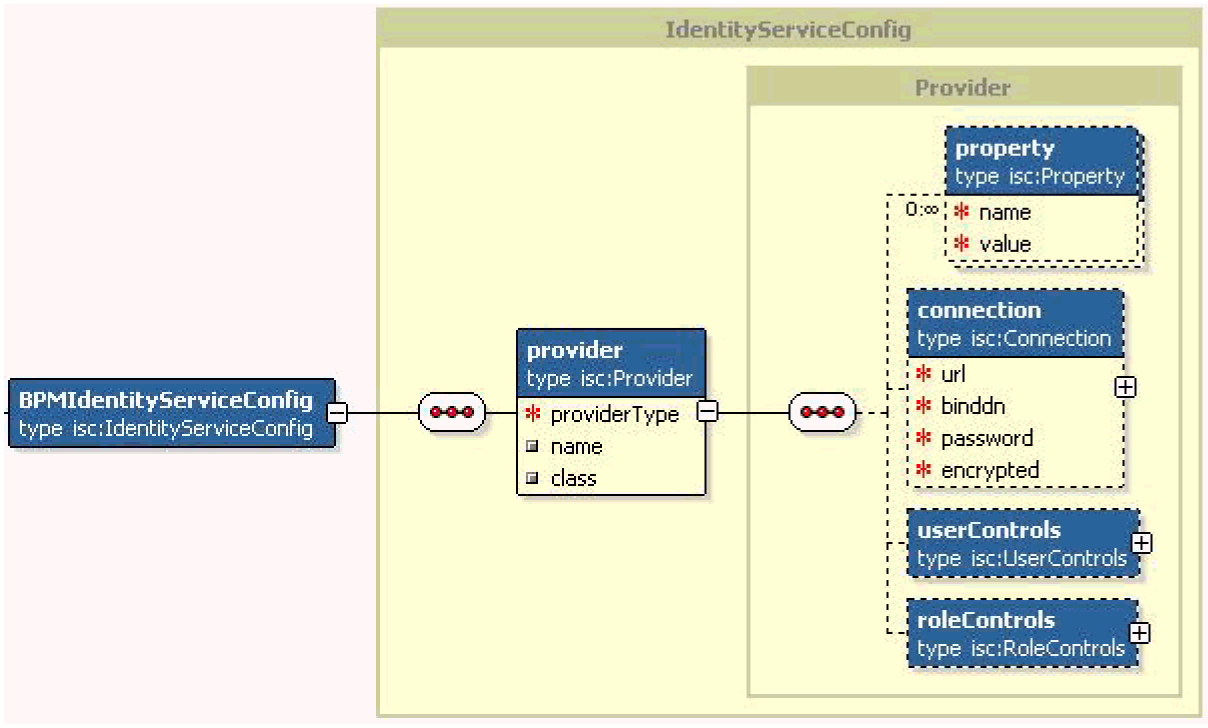
Oracle_Home`\integration\orabpel\system\services\config`

The XML schema for the `is_config.xml` file is stored in

Oracle_Home`\integration\orabpel\system\xmllib\workflow\xsd`

Figure 16–47 shows the structure of the `BPMIdentityService` configuration.

Figure 16–47 BPMIdentityService Configuration



The identity service configuration file (as defined by `is_config.xsd`) consists of a root element `BPMIdentityServiceConfig`, which can have only one provider. As discussed previously, identity service supports the following main plug-in types: JAZN provider, third-party LDAP directories, or custom repository plug-ins.

The provider element specifies the `providerType`, which can be `JAZN`, `LDAP`, or `CUSTOM`, provider name (optional), and any provider-specific properties.

For example, in the case of the JAXN XML provider, you must set the `providerType` attribute to `JAZN` and specify the value of the `userPropertiesFile` attribute. See "Configuration for the XML-Based JAZN Provider" on page 16-83 for more information about `userPropertiesFile`.

Similarly, if you use a custom plug-in to the identity service, you must set the `providerType` attribute to `CUSTOM`. You then specify the class name for custom identity service plug-in implementation, as follows:

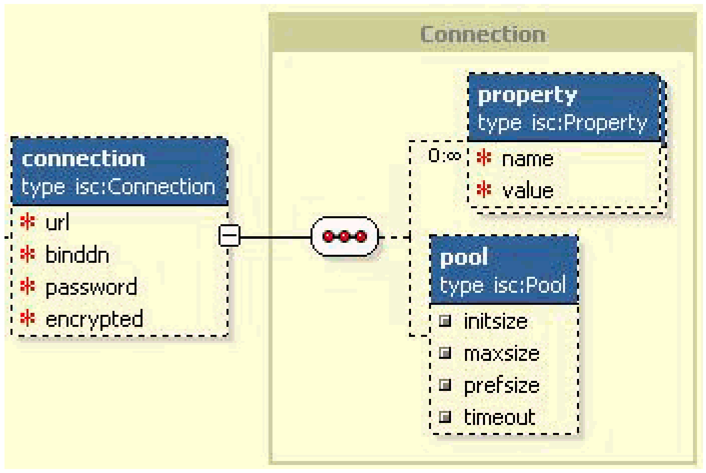
```
<BPMIdentityServiceConfig
mlns="http://www.oracle.com/pcbpel/identityservice/isconfig" >
  <provider providerType="CUSTOM"
    name="CustomPlugIn"
    class="package.name.CustomIdentityService"
  </provider>
</BPMIdentityServiceConfig>
```

In addition, the provider can define the following optional parameters in the configuration file. Most of these parameters apply to JAZN-based or LDAP-based providers, but can be used by custom providers also.

connection Element The `connection` element is used to specify the URL, admin username (`binddn`- *bind as this Distinguished name*), the credential (`password`) for the LDAP or RDBMS connection used by the identity service, and a Boolean flag (`encrypted`) to specify that the password is either in plain text or is encrypted.

Identity service overwrites the `is_config.xml` file after reading the configuration and encrypts the user password if it finds the password in plain text. Figure 16–48 shows the structure of the connection configuration.

Figure 16–48 connection Configuration



The connection can specify connection pool properties by setting the following attributes on the pool element:

- `initsize`—initial size of the connection pool
- `maxsize`—maximum size of the connection pool
- `prefsize`— preferred size of the pool
- `timeout`—time after which the connection is released if there is no activity (in seconds)

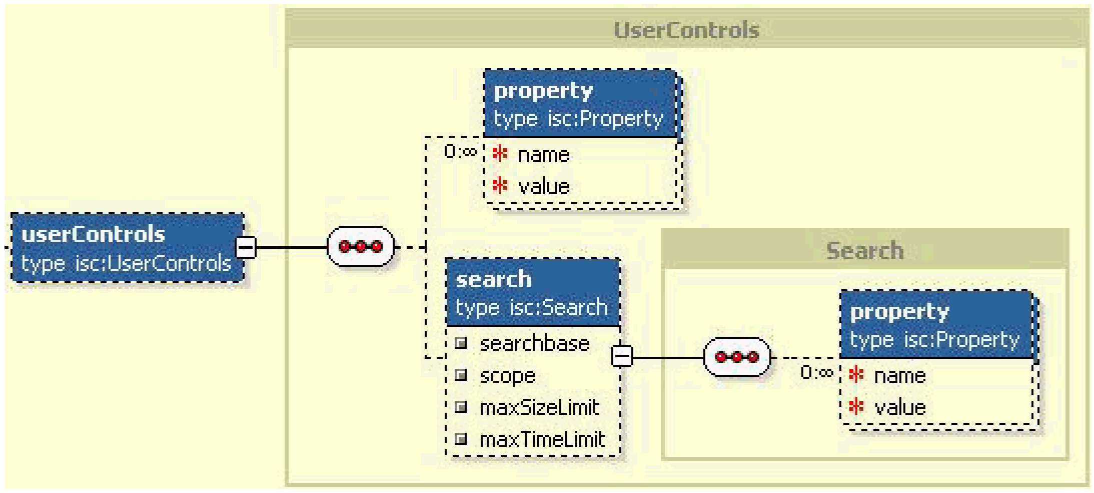
The LDAP plug-in for identity service uses the following default values:

- `initsize="2"`
- `maxsize="25"`
- `prefsize="10"`
- `timeout="60"`

If you are using a custom identity service plug-in, you can also specify any additional connection-specific properties as name-value pairs.

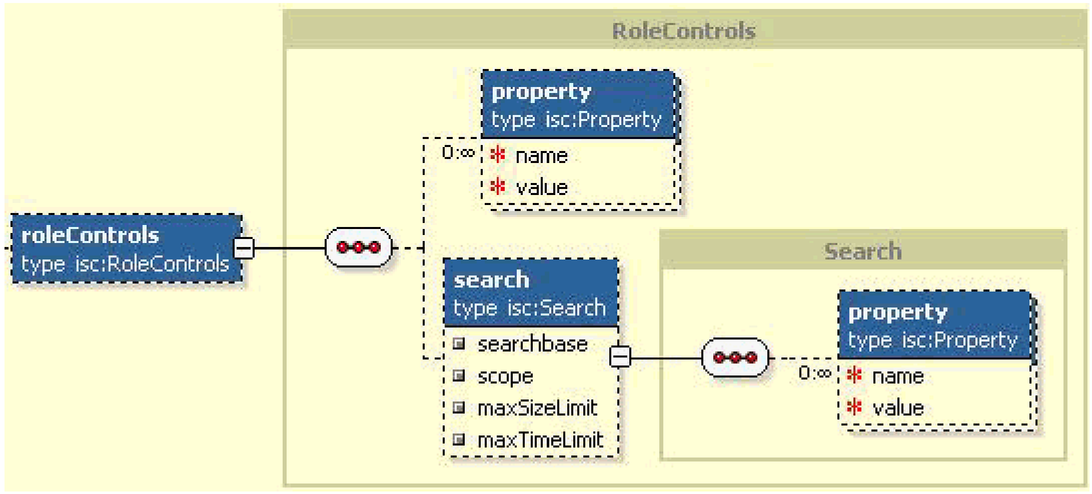
userControls and roleControls Elements The `userControls` element is used to define user controls and to restrict the LDAP user search. Figure 16–49 shows the structure of the `userControls` element.

Figure 16–49 userControls Element



The `roleControls` element is used to define role controls and restrict the LDAP role search. Figure 16–50 shows the structure of the `roleControls` element.

Figure 16–50 roleControls Element



Both `userControls` and `roleControls` can have a search element that has the following optional attributes:

- `searchbase`—a list of LDAP entries, the distinguished names (DNs) of user or group containers.
- `maxSizeLimit`—the maximum number of elements that are fetched from LDAP during a search operation
- `maxTimeLimit`—the maximum time to wait to retrieve elements from an LDAP search
- `scope`—determines the search level. The value can be `onelevel`, in which the search descends one level from the supplied DN or `subtree`, in which the search descends the hierarchy from the DN to the lowest level in the tree.

By default, the LDAP provider for identity service uses the following values: `maxSizeLimit ="1000",maxTimeLimit ="120"` (sec), and `scope="subtree"`.

provider Element The `provider` element enables specifying additional `property` elements, which can be used by custom plug-ins. An example follows:

```
<BPMIdentityServiceConfig
xmlns="http://www.oracle.com/pcbpel/identityservice/isconfig" >
  <provider providerType="CUSTOM" name="db2"
      class="package.name.CustomIdentityService"
    <property name="customProperty" value="customValue" />
  </provider>
</BPMIdentityServiceConfig>
```

In addition, the property element can be defined as part of any of the other elements (`userControls`, `searchControls`, `search`, and so on) in the configuration file.

Configuration for the XML-Based JAZN Provider

The JAZN element `jazn provider="XML" location="./jazn-data.xml"/` is in

```
Oracle_
Home\integration\orabpel\system\appserver\oc4j\j2ee\home\application-deployments\h
w-services\orion-application.xml
```

and

```
Oracle_Home\integration\orabpel\system\appserver\oc4j\j2ee\home\config\jazn.xml
```

The identity service configuration file must specify the `userPropertiesFile` property and provide the value of the file name where all user properties are stored:

```
<IdentityServiceConfig
          xmlns="http://www.oracle.com/pcbpel/identityservice/isconfig"
          xmlns:isc="http://www.oracle.com/pcbpel/identityservice/isconfig"
          providerType="jazn">
  <provider name="xml"
    <property name="userPropertiesFile" value="users-properties.xml"/>
  </provider>
</IdentityServiceConfig>
```

Note that the `users-properties.xml` file from that example stores all extended user's properties. This is not required for JAZN authorization or authentication. However, the BPEL identity service requires this file to get contact details and the organizational hierarchy for users. If this file is not created, then certain workflow functionality such as notifications, manager views, or task escalation may not work. By default the identity service looks for `users-properties.xml` in the Oracle BPEL Process Manager classpath. The Oracle Universal Installer stores the default `users-properties.xml` in

```
Oracle_Home\integration\orabpel\system\services\config
```

> **See Also:** JAZN documentation for how to configure the middle tier to use the XML-based JAZN provider

Configuration for the LDAP-Based JAZN Provider (OID)

You configure the LDAP-based JAZN provider (OID) in two steps:

- OID Configuration
- Middle-Tier Configuration

OID Configuration Use the OID Migration Tool to load Oracle BPEL Process Manager users and roles into OID. The OID Migration Tool can produce LDIF files for system and demo users, which are suitable for loading into a directory server by using standard command-line tools. The input to this tool is a pseudo-LDIF file containing substitution variables. The tool is called ldifmigrator and is found in `Oracle_Home/bin`. (See *Oracle Identity Management Integration Guide* for more information.)

The following predefined substitution variables are used in the file:

Substitution Variable	Value	Description
`%s_UserContainerDN%`	Distinguished name of the entry under which all users are added.	This is assigned the value of the attribute: `orclCommonUserSearchBase` from the entry `cn=Common,cn=Products` under the realm-specific Oracle context.
`%s_GroupContainerDN%`	Distinguished name of the entry under which all public groups are supposed to be added.	This is assigned the value of the attribute: `orclCommonGroupSearchBase` from the entry `cn=Common,cn=Products` under the realm-specific Oracle context.

You can use the command-line variable `s_UserCommonNamingAttribute` with the migration tool, which substitutes all occurrences of it with the value provided in the command-line.

For example:

```
$ldifmigrator "input_file=system-oid.sbs" "output_file=system-oid.ldif" -lookup
"host=ldap.acme.com" "port=389" "subscriber=acme" "s_UserCommonNamingAttribute=cn"
dn="cn=admin" password=welcome

$ldifmigrator "input_file=demo-oid.sbs" "output_file=demo-oid.ldif"
-lookup dn="cn=admin" password=welcome "host=ldap.acme.com" "subscriber=acme"
"s_UserCommonNamingAttribute=cn"
```

where the `subscriber` is the JAZN realm and its name is `acme`, and `cn` is used to construct the user's DN.

The `system-oid.ldif` file can be loaded to OID with the `ldapadd` utility. Optionally, you can load `demo-oid.ldif` with the `ldapmodify` command.

For example:

```
$ldapadd -c -h ldap.acme.com -p 389 -D "cn=admin" -w welcome -f system-oid.ldif
$ldapmodify -c -h ldap.acme.com -p 389 -D "cn=admin" -w welcome -f demo-oid.ldif
```

See Also: *Oracle Identity Management Application Developer's Guide* for information about the `ldapadd` and `ldapmodify` commands

---

Note: By default, when a user enters a search request, OID searches based on the `cn`, `firstname`, `lastname`, and `email` attributes. You can customize the attributes that can be searchable. The user manager attribute from `inetOrgPerson objectClass` should be searchable to allow workflow escalation. Use Oracle Delegated Administration tools to set it up. The recommended searchable attribute list is `cn`, `sn`, `givenName`, `uid`, `manager`, `title`, `mail`, and `telephoneNumber`.

See *Oracle Identity Management Guide to Delegated Administration* for more information.

Middle-Tier Configuration Middle-tier configuration consists of the following steps:

- Configuring the middle tier to use the LDAP-based JAZN provider

  In general, a JAZN XML-based definition should be commented and a new LDAP-based provider definition is specified in both files:

  – *`Oracle_ Home`*`\integration\orabpel\system\appserver\oc4j\j2ee\home\config\jazn.xml`

    ```
    <!-- jazn provider="XML" location="./jazn-data.xml"/ -->
    <jazn provider="LDAP" location="ldap://host:port" default-realm="us" >
      <property name="ldap.user" value="cn=orcladmin" />
      <property name="ldap.password" value="!welcome1" />
    </jazn>
    ```

    where `us` is a default realm name for this example.

  – *`Oracle_ Home`*`\integration\orabpel\system\appserver\oc4j\j2ee\home\application-deployments\hw-services\orion-application.xml`

    ```
    <!-- jazn provider="XML" location="./jazn-data.xml"/ -->
    <jazn provider="LDAP" location="ldap://host:port" default-realm="us" />
    ```

    where `us` is a default realm name for this example.

    Do not place user and password information in `orion-application.xml`.

    Also add the following directive to `Oracle_ Home\integration\orabpel\system\appserver\oc4j\j2ee\home\config\application.xml` to allow password indirection management:

    ```
    <password-manager>
      <jazn provider="XML" location="./jazn-data.xml" />
    </password-manager>
    ```

    See Also:

    - Oracle JAAS Provider (JAZN) configuration documentation
    - *Oracle Application Server Security Guide*

- Configuring identity service configuration file

  Open `is_config.xml` in

  `Oracle_Home``\integration\orabpel\system\services\config`

  and define OID provider properties:

```
<BPMIdentityServiceConfig
xmlns="http://www.oracle.com/pcbpel/identityservice/isconfig">
  <provider providerType="JAZN" name="oid" >
    <connection url="ldap://host:port"
      binddn="cn=orcladmin"
      password="welcome1" encrypted="false" />
  </provider>
</BPMIdentityServiceConfig>
```

The `providerType` must point to the JAZN mode. The provider must contain connection specifications to define OID location, admin user, password and encrypted flag.

Note: Setting the `credentials` element as follows enables you to use clear (readable) passwords in the `jazn-data.xml` file the first time:

- `<credentials>!welcome</credentials>`
- `<credentials>!welcome</credentials>`

This enables the administrator to edit `jazn-data.xml` directly with a text editor. When the file is read and persistence occurs, the password in `jazn-data.xml` is obfuscated and becomes unreadable.

Configuration for a Third-Party LDAP Server

Note the following considerations when using a third-party LDAP server:

- The third-party LDAP servers must be configured to use standard `objectClasses`: `top`, `person`, `organizationalPerson`, `inetOrgPerson`, `groupOfUniqueNames`. If Microsoft Active directory is the third-party LDAP directory provider, then the following `objectClasses` are used: `top`, `person`, `organizationalPerson`, `user`, `group`.

  Usually LDAP servers predefine the list of searchable attributes based on the `cn`, `firstname`, `lastname`, and `email` attributes. You can customize the attributes that can be searchable. The user manager attribute from `inetOrgPerson objectClass` should be searchable to allow workflow escalation. See the documentation for the third-party LDAP server you are using for how to set up the searchable attribute.

  The recommended searchable attribute list is `cn`, `sn`, `givenName`, `uid`, `manager`, `title`, `mail`, and `telephoneNumber`.

- When you seed Oracle BPEL Process Manager users and roles into the LDAP server, the process assumes that the users' and groups' container is created in LDAP.

  To create system and optionally demo ldif files, open the given template files, `system-ldap.sbs` and `demo-ldap.sbs` in

  ```
  Oracle_Home\integration\orabpel\system\services\config\ldap
  ```

  Replace the substitution variables with the appropriate values, as shown in the following example:

Substitution Variable	Replace With Value
`%s_UserCommonNamingAttribute%`	`cn`

Substitution Variable	Replace With Value
%s_UserContainerDN%	ou=People,dc=ldap,dc=acme,dc=com
%s_GroupContainerDN%	ou=Groups,dc=ldap,dc=acme,dc=com

where

- %s_UserContainerDN% with DN value of the entry under which all users are supposed to be added. The users container with dn: ou=People,dc=ldap,dc=acme,dc=com is used in this example.
- %s_GroupContainerDN% with DN value of the entry under which all public groups are supposed to be added. The groups container with dn: ou=Groups,dc=ldap,dc=acme,dc=com is used in this example.
- %s_UserCommonNamingAttribute% with the value used to construct the user's DN. In this example the cn value is used.

Store changes in the system-ldap.ldif and demo-ldap.ldif files. Then load the system-ldap.ldif file to the LDAP server by using the ldapadd utility. Optionally, load demo-ldap.ldif with the ldapmodify utility.

For example:

```
$ldapadd -c -h ldap.acme.com -p 389 -D "cn=admin" -w welcome -f system-oid.ldif
$ldapmodify -c -h ldap.acme.com -p 389 -D "cn=admin" -w welcome -f
demo-oid.ldif
```

See the documentation for the third-party LDAP server you are using for information about the ldapadd and ldapmodify commands.

- The identity service third-party LDAP provider must specify connection, userControls, and roleControls elements in the identity service configuration file.

  Identity service third-party LDAP provider implementation defines a set of user search properties that must be configured:

  - nameattribute—the name of the LDAP attribute that uniquely identifies the name of the user. In Sun Directory Server, it is uid; in Active Directory, it is user.
  - objectClass—the LDAP schema object class used to represent a user. In Sun Directory Server, it is inetOrgPerson.

  And set of role search properties:

  - nameattribute—the name of the LDAP attribute that uniquely identifies the name of the role. In Sun Directory Server, it is uniqueMember; in Active Directory, it is member.
  - objectclass—the LDAP schema object class that is used to represent a group. In Sun Directory Server, it is groupOfUniqueNames. In Active Directory, it is group.
  - membershipsearchscope—specifies how deep in the LDAP directory tree to search for role membership. Supported values: onelevel or subtree.
  - memberattribute—The attribute of a static LDAP group object specifying the distinguished names (DNs) of the members of the group. In Sun Directory Server, it is uniqueMember; in Active Directory, it is member.

Both `userControls` and `roleControl` must define a search element with the `searchbase` attribute.

The `searchbase` attribute of the `userControls` search element is a space-separated list of DNs in the LDAP directory that contains users; for example, `cn=users,dc=us,dc=abc,dc=com`.

The `searchbase` attribute of the `roleControls` search element is a space-separated list of DNs in the LDAP directory that contains roles; for example, `cn=Groups,dc=us,dc=abc,dc=com`

An example of the LDAP server Sun Directory Server configuration follows:

```
<BPMIdentityServiceConfig
xmlns="http://www.oracle.com/pcbpel/identityservice/isconfig">
  <provider providerType="LDAP" name="iplanet"
    <property name="realmName" value="iPlanetRealm"/>
      <connection url="ldap://host:port"
         binddn="uid=admin,ou=administrators,ou=topologymanagement,o=netscaperoot"
         password="welcome" encrypted="false" >
        <pool initsize="2" maxsize="25" prefsize="10" timeout="60"/>
      </connection>
    <userControls >
      <property name="nameattribute" value="uid"/>
      <property name="objectclass" value="inetOrgPerson"/>
        <search searchbase="ou=People,dc=us,dc=oracle,dc=com"
maxSizeLimit="1000" maxTimeLimit="120" scope="onelevel" />
    </userControls>

    <roleControls >
      <property name="nameattribute" value="cn"/>
      <property name="objectclass" value="groupOfUniqueNames"/>
      <property name="membershipsearchscope" value="onelevel"/>
      <property name="memberattribute" value="uniquemember"/>
      <search searchbase="ou=Groups,dc=us,dc=oracle,dc=com"
             maxSizeLimit="1000" maxTimeLimit="120" scope="onelevel" />
    </roleControls>
  </provider>
</BPMIdentityServiceConfig>
```

An example for Microsoft Active Directory follows:

```
<BPMIdentityServiceConfig
xmlns="http://www.oracle.com/pcbpel/identityservice/isconfig">
  <provider providerType="CUSTOM" name="Active Directory"
    <property name="realmName" value="ActiveDirectoryRealm"/>
    <connection url="ldap://host:port"
          binddn="cn=administrator,cn=Users,dc=us,dc=oracle,dc=com"
          password="welcome" encrypted="false" />
    <userControls >
      <property name="nameattribute" value="cn"/>
      <property name="objectclass" value="user"/>
      <search searchbase="cn=Users,dc=us,dc=oracle, dc=com",
            maxSizeLimit="1000" maxTimeLimit="120 " scope="onelevel" />
    </userControls>

    <roleControls >
      <property name="nameattribute" value="cn"/>
      <property name="objectclass" value="group"/>
      <property name="membershipsearchscope" value="onelevel"/>
      <property name="memberattribute" value="member"/>
      <search searchbase="cn=Users,dc=us,dc=oracle,dc=com"
```

```
          maxSizeLimit="1000" maxTimeLimit="120" scope="onelevel" />
    </roleControls>
  </provider>
</BPMIdentityServiceConfig>
```

Configuration for CUSTOM User Repository Plug-ins

The following example shows how to use configuration properties to configure custom plug-ins. In this example, `CustomIdentityService` class is used to demonstrate custom repository plug-ins. This class implements the `BPMIdentityService` interface.

```
<BPMIdentityServiceConfig
mlns="http://www.oracle.com/pcbpel/identityservice/isconfig">
  <provider providerType="CUSTOM" name="CustomProvider"
        class="custompakage.CustomIdentityService">
    <property name="realmName" value="CustomRealm"/>
    <property name="CustomProviderProperty1" value="CustomProviderValue1"/>
    <property name="CustomProviderProperty2" value="CustomProviderValue2"/>
    <connection url="ldap://host:port"
             binddn="uid=admin password="welcome" encrypted="false" >
      <property name="CustomConnProperty1" value="CustomConnValue1"/>
      <property name="CustomConnProperty2" value="CustomConnValue2"/>
    </connection>

    <userControls>
      <property name="CustomControlsProperty1" value="CustomControlsValue1"/>
      <property name="CustomControlsProperty2" value="CustomControlsValue2"/>
      <search searchbase="ou=UserContainer,dc=us,dc=oracle,dc=com">
        <property name="CustomSearchProperty1" value="CustomSearchValue1"/>
        <property name="CustomSearchProperty2" value="CustomSearchValue2"/>
      </search>
    </userControls>

    <roleControls >
      <property name="CustomControlsProperty1" value="CustomControlsValue1"/>
      <property name="CustomControlsProperty2" value="CustomControlsValue2"/>
      <search searchbase="ou=GroupContainer,dc=us,dc=oracle,dc=com">
        <property name="CustomSearchProperty1" value="CustomSearchValue1"/>
        <property name="CustomSearchProperty2" value="CustomSearchValue2"/>
      </search>
    </roleControls>
  </provider>
</BPMIdentityServiceConfig>
```

In addition to existing provider properties, you can define custom property elements that can be added to `provider`, `connection`, `userControls`, `roleControls`, and `search` elements in the configuration file to extend provider definitions.

Workflow-Related XPath Extension Functions

XPath extension functions mimic XPath 2.0 standards. The following extension functions are available.

ora:lookupUser(userId)

This extension function is used to look up a user. The function returns an XML element of complex type as defined by the schema in `LocalIdentityService.xsd` at http://*hostname:port*/orabpel/xmllib/workflow/

The following code example demonstrates how to use the extension function.

```
<process...
    xmlns:idservice= http://xmlns.oracle.com/pcbpel/identityservice/local
...
<variables>
...
    <variable name="user" element="idservice:user"/>
</variables>
  <sequence>
...
    <assign name="lookupUser">
      <!-- get the user-->
      <copy>
        <from expression="ora:lookupUser(bpws:getVariableData('input', 'payload',
'/tns:input/tns:userId'))"/>
          <to variable="user"/>
      </copy>
    </assign>
...
  </sequence>
</process>
```

If `userId` is not a valid user, the function returns null.

ora:lookupGroup(groupId)

This extension function is used to look up a group. The function returns an XML element of complex type as defined by the schema in `LocalIdentityService.xsd` at

http://*hostname:port*/orabpel/xmllib/workflow/

The following code example demonstrates how to use the extension function.

```
<process …
    xmlns:idservice= http://xmlns.oracle.com/pcbpel/identityservice/local
...
<variables>
...
    <variable name="group" element="idservice:group"/>
</variables>
  <sequence>
...
    <assign name="lookupGroup">
      <!-- get the group-->
      <copy>
        <from expression="ora:lookupGroup(bpws:getVariableData('input', 'payload',
'/tns:input/tns:groupId'))"/>
          <to variable="group"/>
      </copy>
    </assign>
...
  </sequence>
</process>
```

If `groupId` is not a valid user, the function returns null.

ora:getUserProperty(userId, attributeName)

This function can be used to get any property of the user. The arguments to the function are as follows:

- `userId`—String or element containing the user whose attribute is to be retrieved
- `attributeName`—String or element containing the name of the user attribute. The attribute name is one of the following values:
  - `givenName`
  - `middleName`
  - `sn`
  - `displayName`
  - `mail`
  - `telephoneNumber`
  - `homephone`
  - `mobile`
  - `facsimileTelephoneNumber`
  - `pager`
  - `preferredLanguage`
  - `preferredLanguage`
  - `manager`

If the user with the given `userId` does not exist, the function returns null. If the user does not have the given property, or the value for the property is empty, then the function returns the string `undefined`.

ora:getGroupProperty(groupId, attributeName)

This function can be used to get any property of the group. The arguments to the function are as follows:

- `groupId`—String or element containing the group whose attribute is to be retrieved
- `attributeName`—String or element containing the name of the group attribute. The attribute name should be one of the following values:
  - `displayName`
  - `mail`

If the group with the given `groupId` does not exist, the function returns null. If the group does not have the given property, or the value for the property is empty, then the function returns the string `undefined`.

ora:getManager(userId)

This extension function is used to get the manager of a user identified by the `userId`. This function returns a string identifying the manager of the user. If the user is not valid, or if the user does not have a manager, then the function returns null.

ora:getReportees(userId)

This function gets the direct reportees of a user identified by the `userId`. The function returns a list of nodes. Each node in the list is called `user`. The namespace URI of the node is

```
http://oracle.tip.pc.services.identity/RemoteIdentityService.xsd
```

If the user does not exist, then the function returns null.

ora:getUsersInGroup(groupId)

This function gets the users in a group. The function returns a list of nodes. Each node in the list is called `user`. The namespace URI of the node is

```
http://oracle.tip.pc.services.identity/RemoteIdentityService.xsd
```

If the group does not exist, then the function returns null.

ora:getUserRoles(userId, roleType, direct)

This function gets the user roles. The function returns a list of objects, either role objects or group objects, depending on the `roleType`. The arguments to the function are as follows:

- `userId`—String or element containing the user whose roles are to be retrieved.
- `roleType`—The role type, which has one of three values: `ApplicationRole`, `EnterpriseRole`, or `AnyRole`.
- `direct`—String or element indicating if direct or indirect roles are to be fetched. This is optional and, if not specified, only direct roles are fetched. It is either `xsd:boolean` or `string true/false`.

The function returns a list of nodes. Each node in the list is called `group` or `role`, depending on the `roleType`. The namespace URI of the node is

```
http://oracle.tip.pc.services.identity/RemoteIdentityService.xsd
```

ora:isUserInRole(userId, roleName)

This function verifies if a user identified by the `userId` has a given role identified by `roleName`. The function returns a Boolean `true` or `false`.

ora:getNumberOfTaskApprovals(taskId)

This extension function returns the number of times (an integer) that a task identified by the given `taskId` is approved (approved in a generic sense, not the outcome `approve`) by users. The function returns null if there is no task with the given `taskId`.

ora:getPreviousTaskApprover(taskId)

This extension function returns the previous user who approved (approved in a generic sense, not the outcome `approve`) a task identified by the given `taskId`. The function returns a string `userId` that identifies the previous approver. The function returns null if there is no task with the given `taskId`. The return of this function can be used to get the title of the previous approver, for example, as follows:

```
ora:getUserProperty(ora:getPreviousTaskApprover(tasked), ‘title’)
```

ora:getTaskAttachmentsCount(taskId)

This function returns the number of attachments (an integer) in a task that is identified by the given `taskId`. The function returns null if there is no task with the given `taskId`.

ora:getTaskAttachmentByIndex(taskId, attachmentIndex)

This function returns the attachment for the task identified by the given `taskId` at the given `attachmentIndex`. The function returns an element of the type `{http://xmlns.oracle.com/pcbpel/taskservice/task}attachment`. This type is defined in `WorkflowTask.xsd`. Each attachment has either content or a URI. The content, if any, in the element returned by the function is a Base64-encoded string.

The following BPEL code example demonstrates how to use the `getTaskAttachmentsCount` and `getTaskAttachmentByIndex` functions. The BPEL shown gets all the attachments in the task and writes to a file using the file adapter.

```
<process .......
    xmlns:ora="http://schemas.oracle.com/xpath/extension"
    xmlns:task="http://xmlns.oracle.com/pcbpel/taskservice/task">

...

<variables>
...
  <variable name="SimpleWorkflowVar1" element="task:task"/>
  <variable name="TaskAttachment" element="task:attachment"/>
</variables>

<sequence name="main">
...

  <assign name="Assign_1">
    <copy>
      <from expression="number(0)"/>
      <to variable="AttachmentIndex"/>
    </copy>
  </assign>
  <while name="While_1"
condition="bpws:getVariableData
('AttachmentIndex') &lt; ora:getTaskAttachmentsCount
(bpws:getVariableData('SimpleWorkflowVar1','/task:task/task:taskId'))">
    <sequence name="Sequence_1">
      <assign name="Assign_2">
        <copy>
          <from expression="ora:getTaskAttachmentByIndex(bpws:getVariableData
('SimpleWorkflowVar1','/task:task/task:taskId'),(bpws:getVariableData
('AttachmentIndex') + number(1)))"/>
          <to variable="TaskAttachment" query="/task:attachment"/>
        </copy>
          <copy>
            <from expression="bpws:getVariableData('AttachmentIndex') +
number(1)"/>
            <to variable="AttachmentIndex"/>
          </copy>
          <copy>
            <from variable="TaskAttachment"
        query="/task:attachment/task:content"/>
            <to variable="Invoke_1_Write_InputVariable" part="opaque"
```

```
query="/ns2:opaqueElement"/>
          </copy>
        </assign>
        <invoke name="Invoke_1" partnerLink="File" portType="ns1:Write_ptt"
operation="Write" inputVariable="Invoke_1_Write_InputVariable"/>
      </sequence>
    </while>
...
  </sequence>
</process>
```

ora:getTaskAttachmentByName(taskId, attachmentName)

This function returns the attachment for the task identified by the given `taskId` at the given `attachmentName`. The function returns an element of the type `{http://xmlns.oracle.com/pcbpel/taskservice/task}attachment`. This type is defined in `WorkflowTask.xsd`. Each attachment has either content or a URI. The content, if any, in the element returned by the function is a Base64-encoded string.

The following BPEL code example demonstrates how to use the `getTaskAttachmentByName` function. The BPEL shown gets the attachment with the name `utplan.doc` and writes to a file using the file adapter.

```
<process ........
      xmlns:ora="http://schemas.oracle.com/xpath/extension"
      xmlns:task="http://xmlns.oracle.com/pcbpel/taskservice/task">

...

  <variables>
  ...
  <variable name="SimpleWorkflowVar1" element="task:task"/>
  <variable name="TaskAttachment" element="task:attachment"/>
  </variables>

<sequence name="main">
...

    <assign name="Assign_3">
      <copy>
        <from expression="ora:getTaskAttachmentByName(bpws:getVariableData
('SimpleWorkflowVar1','/task:task/task:taskId'), string('utplan.doc'))"/>
        <to variable="TaskAttachment" query="/task:attachment"/>
      </copy>
      <copy>
        <from variable="TaskAttachment" query="/task:attachment/task:content"/>
        <to variable="Invoke_2_Write_InputVariable" part="opaque"
query="/ns2:opaqueElement"/>
      </copy>
    </assign>
    ...
  </sequence>
</process>
```

ora:getTaskAttachmentContents(taskId, attachmentName)

This function returns the attachment for the task identified by the given `taskId` at the given `attachmentName`. The function returns a Base64-encoded string of either the attachment content or the URL (each attachment has either a content or a URI). The difference between this function and the `getTaskAttachmentByName` function is that the `getTaskAttachmentByName` function returns a complex element, whereas this function returns the contents only. The following BPEL code example demonstrates how to use the `getTaskAttachmentContents` functions. The BPEL shown gets the attachment with the name `utplan.doc` and writes to a file using the file adapter.

```
<process ........
        xmlns:ora="http://schemas.oracle.com/xpath/extension"
        xmlns:task="http://xmlns.oracle.com/pcbpel/taskservice/task">

...

  <variables>
   ...
    <variable name="SimpleWorkflowVar1" element="task:task"/>
  </variables>

  <sequence name="main">
   ...

    <assign name="Assign_4">
      <copy>
        <from expression="ora:getTaskAttachmentContents(bpws:getVariableData
('SimpleWorkflowVar1','/task:task/task:taskId'), string('utplan.doc'))"/>
        <to variable="Invoke_2_Write_InputVariable" part="opaque"
query="/ns2:opaqueElement"/>
      </copy>
    </assign>
    <invoke name="Invoke_3" partnerLink="File" portType="ns1:Write_ptt"
operation="Write" inputVariable="Invoke_2_Write_InputVariable"/>
  ...
  </sequence>
</process>
```

orcl:get-localized-string(resourceURL,resourceLocation,resourceBundleName,language,country,variant,messageKey)

This extension function can be used to get localized messages for notifications for internationalization.

orcl:format-string(string,string,string,string, ...)

The `format-string` extension function can be used to format strings during construction of messages for notifications. This can be useful if the localized message must be formatted with data from the payload.

Approver Functions

In a sequential workflow scenario, the users or groups to whom the task is routed are captured using functions. The function is stored in the `approver` element of the task object. These functions are evaluated at run time to determine the next approvers.

Approver Function Syntax

The approver functions are governed by the following grammar.

approverFunction := (managementChainFunction | listFunction | adhocFunction | usersFunction | groupsFunction)*

A comma-separated list of approver functions.

managementChainFunction := managementChain(level, title)

Represents a management chain. The management chain includes users within a number of levels and up to a user whose title is specified. `level` is a required argument and `title` is an optional argument.

listFunction := list(usersFunction*, groupsFunction*, acquiredByFunction)

Represents a single assignment to multiple users or groups. This function can be used to assign the task to a mix of users and groups. Optionally, the function can also specify the acquirer of the task when the task is assigned to a group or multiple users. The `acquiredBy` in this function overwrites the `acquiredBy` in `usersFunction` and `groupsFunction`.

adhocFunction := adhoc()

Allows the current approver to specify who the next approver is. When the user completes the task without specifying the next approvers, the function is no longer evaluated.

usersFunction := users(userId*, acquiredByFunction)

Represents a single task assignment to one or more users. Optionally the function can also specify the acquirer of the task when the task is assigned to multiple users. Each argument in this function must be a valid user id from the user store (OID, LDAP, and so on).

groupsFunction := groups(groupId*, acquiredByFunction)

Represents a single task assignment to one or more groups. Optionally, the function can also specify the acquirer of the task when the task is assigned to groups. Each argument in this function must be a valid group id from the user store (OID, LDAP, and so on).

acquiredByFunction := acquiredBy(userId)

Captures the acquirer when the task is assigned to a set of users or groups. The argument in this function must be a valid user id from the user store (OID, LDAP, and so on).

Where

- `# userId` is a string argument representing the id of the user.
- `groupId` is a string argument representing the id of the group.
- `level` is a number argument that represents the levels in the management chain.
- `title` is a string argument that represents the title of the last user in the management chain.
- All arguments should be wrapped in quotes (" ").
- The arguments are separated by commas.
- The management chain function is always evaluated with respect to the previous approver of the task.

Approver Function Examples

The following examples show how to use the approver functions.

- `managementChain("2", "Manager")`

Routes the task to users in the management chain. The management chain includes users within 2 levels and up to a user whose title is `Manager`.

- `list(users("jcooper", "jstein"), groups("LoanAgentRole"), acquiredBy("jcooper"))`

  Routes the task once to users `jcooper` and `jstein` and group `LoanAgentRole` and also sets `acquiredBy` to `jcooper`.

- `list(users("jcooper", "jstein")), list(groups("LoanAgentRole"))`

  Routes the task to users `jcooper` and `jstein`. When one of those users acquires and acts on the task, routes the task to the group `LoanAgentRole`.

- `adhoc()`

  The task supports adhoc routing and the next users or groups are specified by the current approver of the task.

- `users("jcooper", "jstein")`

  Routes the task once to users `jcooper` and `jstein`.

- `users("jcooper", "jstein", acquiredBy("jcooper"))`

  Routes the task once to users `jcooper` and `jstein`. Also sets `acquiredBy` to `jcooper`.

- `users("jcooper"), users("jstein")`

  Routes the task first to user `jcooper`, and after `jcooper` acts on the task, routes the task to `jstein`.

- `groups("LoanAgentRole", "Supervisor")`

  Routes the task once to the groups `LoanAgentRole` and `Supervisor`.

- `groups("LoanAgentRole"), groups("Supervisor")`

  Routes the task first to the group `LoanAgentRole` and after a user from the `LoanAgentRole` acts on the task, routes the task to the group `Supervisor`.

- `groups("LoanAgentRole", "Supervisor", acquiredBy("jcooper"))`

  Routes the task once to groups `LoanAgentRole` and `Supervisor`. Sets `acquiredBy` to `jcooper`.

- `managementChain("2", "Manager"), groups("LoanAgentRole")`

  Routes the task to users in the management chain. The management chain includes users within two levels and up to a user whose title is `Manager`. After the users in the management chain are exhausted, routes the task to the group `LoanAgentRole`.

- `managementChain(“2”, “Manager”), groups(“LoanAgentRole”), adhoc()`

  Routes the task to users in the management chain. The management chain includes users within two levels and up to a user whose title is `Manager`. After the users in the management chain are exhausted, routes the task to the group `LoanAgentRole`. When a user in the `LoanAgentRole` acquires the task and acts on it, the task can still be routed to other users as specified by the approver of the task.

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 simple workflow using JDeveloper BPEL Designer
- Using the Worklist Application to view and respond to tasks

Prerequisites

This example assumes the following:

- You should be familiar with basic BPEL constructs, including BPEL activities and partner links, and basic XPath functions. Familiarity with JDeveloper BPEL Designer—the Oracle JDeveloper environment for creating and deploying BPEL processes—is also assumed.
- You must change the e-mail address for the user `jstein.` If the XML-based JAZN provider is used, these properties can be changed in

  `Oracle_Home``\integration\orabpel\system\services\config\users-properties.xml`

  The following XML segment from the `users-properties.xml` file shows where the e-mail is configured:

  ```
  <bpm:BPMUser userName="jstein" >
  <bpm:properties>
  <bpm:givenName>John</bpm:givenName>
  <bpm:sn>Steinbeck</bpm:sn>
  <bpm:title>Manager2</bpm:title>
  <bpm:manager>wfaulk</bpm:manager>
  <bpm:mail>user2@dlsun4254.us.oracle.com</bpm:mail>
  <bpm:timeZone>GMT-8</bpm:timeZone>
  <bpm:preferredLanguage>en-US</bpm:preferredLanguage>
  <bpm:orclWorkflowNotificationPref>Mail</bpm:orclWorkflowNotificationPref>
  </bpm:properties>
  </bpm:BPMUser>
  ```

- You must configure the e-mail server settings for the account `Default`, if it is different from the default values. The `Default` account is used to send e-mails. The e-mail server configuration is in

  `Oracle_Home``\integration\orabpel\system\services\config\ns_emails.xml`

  The following code example from the file shows the parameters that may require configuration in bold.

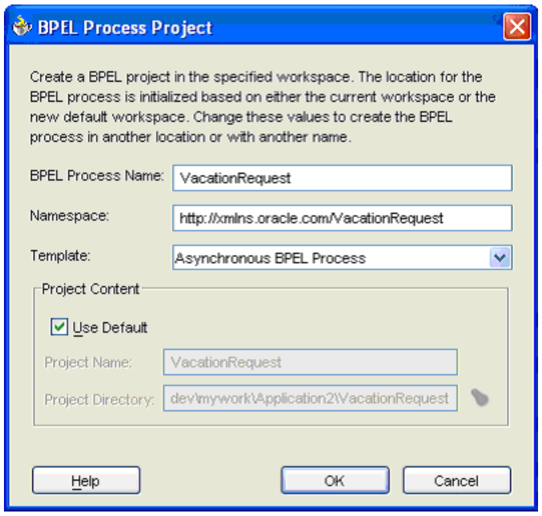
  ```
  <EmailAccount>
  <Name>Default</Name>
  <GeneralSettings>
  <FromName>Oracle BPM</FromName>
  <FromAddress>bpm1@dlsun4254.us.oracle.com</FromAddress>
  </GeneralSettings>
  <OutgoingServerSettings>
  <SMTPHost>dlsun4254.us.oracle.com</SMTPHost>
  <SMTPPort>225</SMTPPort>
  </OutgoingServerSettings>
  <IncomingServerSettings>
  ```

```
<Server>dlsun4254.us.oracle.com</Server>
<Port>2110</Port>
<Protocol>pop3</Protocol>
<UserName>bpm1</UserName>
<Password ns0:encrypted="false"
xmlns:ns0="http://xmlns.oracle.com/ias/pcbpel/NotificationService">welcome</Pas
sword>
<UseSSL>false</UseSSL>
<Folder>Inbox</Folder>
<PollingFrequency>1</PollingFrequency>
<PostReadOperation>
<MarkAsRead/>
</PostReadOperation>
</IncomingServerSettings>
</EmailAccount>
```

- You must restart Oracle BPEL Process Manager after making any of the preceding changes.
- Verify that the task payload is displayed in the Worklist Application using the XSL file `vacationRequest.xsl`. This file is needed to complete the tutorial and is available in the `VacationRequest` sample business process.

Getting Started: Modeling the Vacation Request Process

1. Create a business process called **VacationRequest**.
2. When prompted for a template, select **Asynchronous BPEL Process**.



Create a BPEL project in the specified workspace. The location for the BPEL process is initialized based on either the current workspace or the new default workspace. Change these values to create the BPEL process in another location or with another name.

BPEL Process Name: VacationRequest

Namespace: http://xmlns.oracle.com/VacationRequest

Template: Asynchronous BPEL Process

Project Content

[✓] Use Default

Project Name: VacationRequest

Project Directory: dev\mywork\Application2\VacationRequest

3. After creating the new process, change the message structure of the vacation request in the `VacationRequest.wsdl` to be more relevant to a vacation request process. The message structure by default is

```
<element name="VacationRequestProcessRequest">
  <complexType>
    <sequence>
      <element name="input" type="string"/>
    </sequence>
  </complexType>
</element>
```

Replace the element called `input` with the four elements shown in bold in the following:

```
<element name="VacationRequestProcessRequest">
  <complexType>
    <sequence>
      <element name="creator" type="string"  />
      <element name="fromDate" type="date" />
      <element name="toDate" type="date" />
      <element name="reason" type="string" />
    </sequence>
  </complexType>
</element>
```

4. Copy `vacationRequest.xsl` to the project location and add the file to the project.
5. Drop a **User Task** activity between the **Receive** activity and the **callbackClient** activity.
6. In the Workflow wizard, leave the default selected for **Create New Workflow** and click **Next**.
7. From **Workflow Pattern**, select **Simple Workflow** and click **Next**.
8. Use the autocomplete feature to assign a value to the title. For example, use **Vacation Request** for

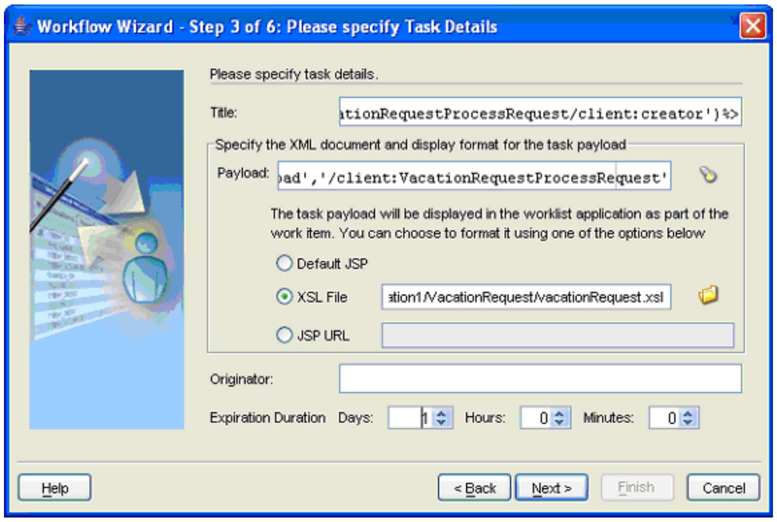
   ```
   <%bpws:getVariableData('inputVariable','payload','/
   client:VacationRequestProcessRequest/client:creator')%>
   ```

9. Use the autocomplete feature to assign the payload as

   ```
   bpws:getVariableData('inputVariable','payload','/
   client:VacationRequestProcessRequest')
   ```

10. Leave **Auto generate JSP form** as the payload display option.

    With this option, a JSP is autogenerated based on the XML type of the payload chosen in the previous step.
11. Leave the **Task Creator** field blank.
12. Set **Expiration Duration Days** to `1`.



Please specify task details.

Title: ationRequestProcessRequest/client:creator')%>

Specify the XML document and display format for the task payload

Payload: ad','/client:VacationRequestProcessRequest'

The task payload will be displayed in the worklist application as part of the work item. You can choose to format it using one of the options below

Default JSP [✓ XSL File] JSP URL

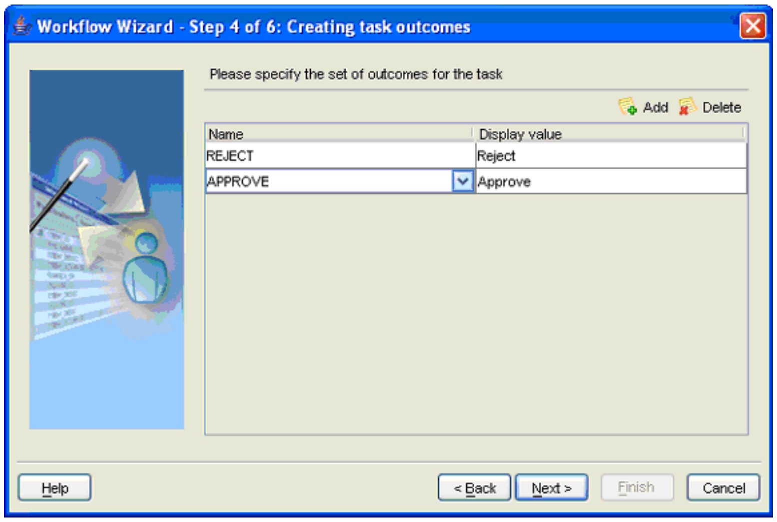
XSL File: ation1/VacationRequest/vacationRequest.xsl

Originator:

Expiration Duration Days: 1 Hours: 0 Minutes: 0

13. Click **Next**.

14. By default, there are two outcomes: accept and reject. Delete **ACCEPT** and add **APPROVE**. Click **Next**.

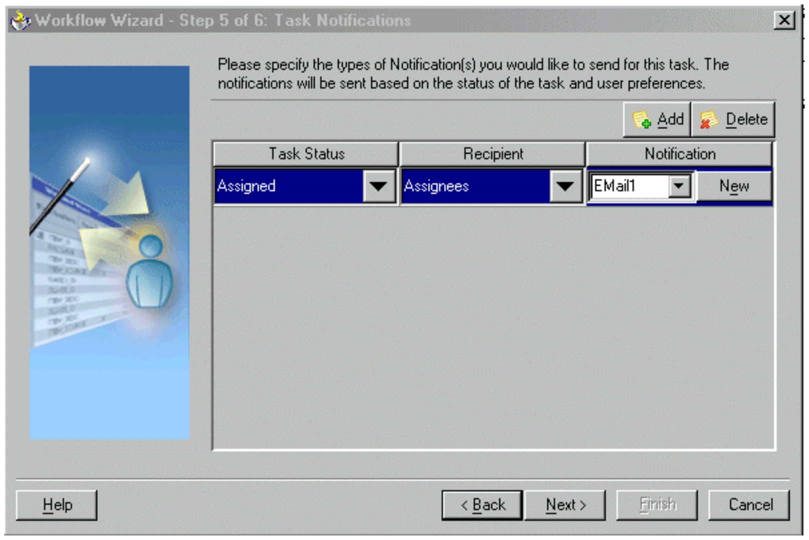


Workflow Wizard - Step 4 of 6: Creating task outcomes

Please specify the set of outcomes for the task

Name	Display value
REJECT	Reject
APPROVE	Approve

15. On the Notification page, select **Assigned** for **Task Status**, **Assignees** for **Recipient**, and **Email** for **Notification**.



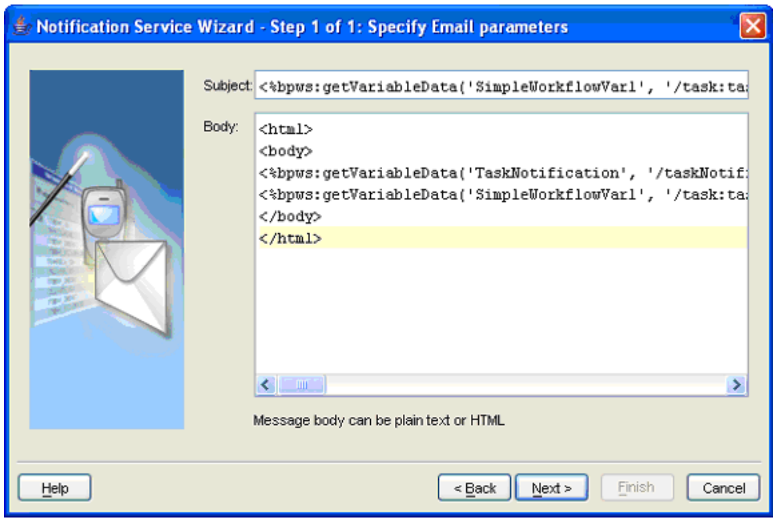
Workflow Wizard - Step 5 of 6: Task Notifications

Please specify the types of Notification(s) you would like to send for this task. The notifications will be sent based on the status of the task and user preferences.

Task Status	Recipient	Notification
Assigned	Assignees	EMail1 New

You can also use the e-mail wizard to change default e-mail content.

16. Click **New** to start the wizard.


Notification Service Wizard - Step 1 of 1: Specify Email parameters
Help
< Back
Next >
Finish
Cancel

Subject: <%bpws:getVariableData('SimpleWorkflowVar1', '/task:ta

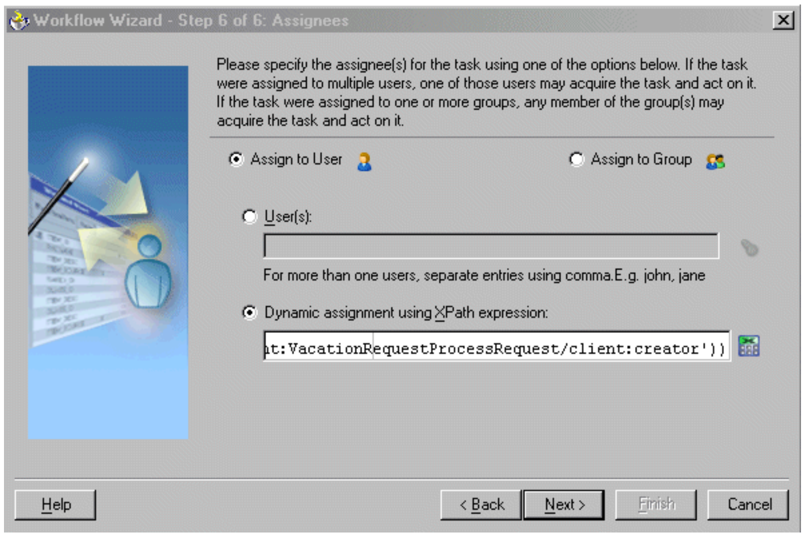
Body:

```
<html>
<body>
<%bpws:getVariableData('TaskNotification', '/taskNotif:
<%bpws:getVariableData('SimpleWorkflowVar1', '/task:ta
</body>
</html>
```

Message body can be plain text or HTML

17. Click **Next**.
18. In the Assignees page, select **Dynamic user assignment using XPath expression**. Use autocomplete to set the assignee to

    ```
    ora:getManager(bpws:getVariableData('inputVariable','payload','/
    client:VacationRequestProcessRequest/client:creator'))
    ```



Please specify the assignee(s) for the task using one of the options below. If the task were assigned to multiple users, one of those users may acquire the task and act on it. If the task were assigned to one or more groups, any member of the group(s) may acquire the task and act on it.

[✓ Assign to User] Assign to Group

User(s):

For more than one users, separate entries using comma.E.g. john, jane

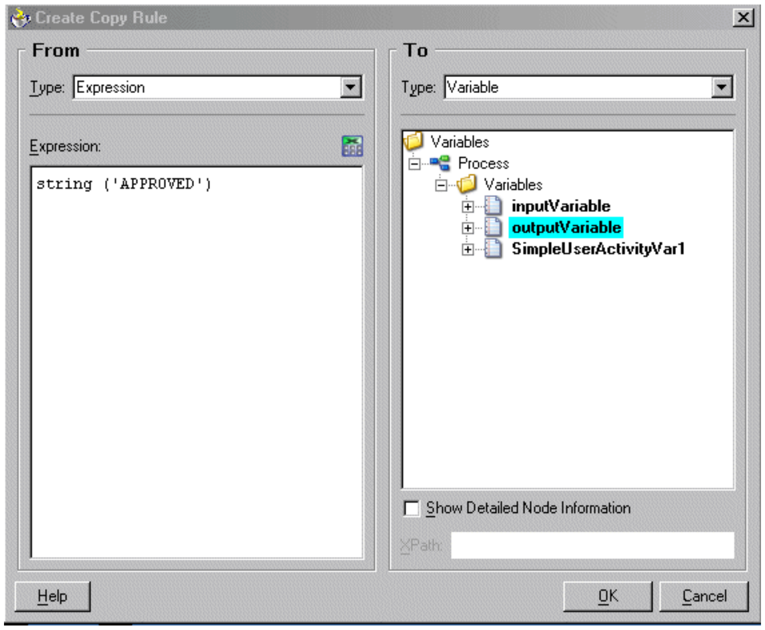
[✓ Dynamic assignment using XPath expression:]

t:VacationRequestProcessRequest/client:creator'))

19. Click **Next**.
20. Click **Finish**.

    You should see a **Scope** and a **Switch** activity. The **Switch** activity has three cases associated with it. Each of the case statements represents the possible outcome specified (approve and reject in this tutorial), plus an otherwise section to represent any other conclusion of the task (errored, expired, and so on). Inside each of the case activities, you can add activities to complete modeling the business process. By default, there is an **Assign** activity named **copyPayloadFromTask** in each of the branches. This **Assign** copies the payload back to its source.
21. Add an **Assign** activity to the case for **Task outcome is APPROVE**.

22. In the Assign window (double-click **Assign** to invoke this window), click the **Copy Rules** tab and click **Create**.
23. In the Create Copy Rule window, do the following:
    a. In the **From** section, click **Expression** and enter `string('APPROVED')`.
    b. In the **To** section, for **Variable**, select **outputVariable**.

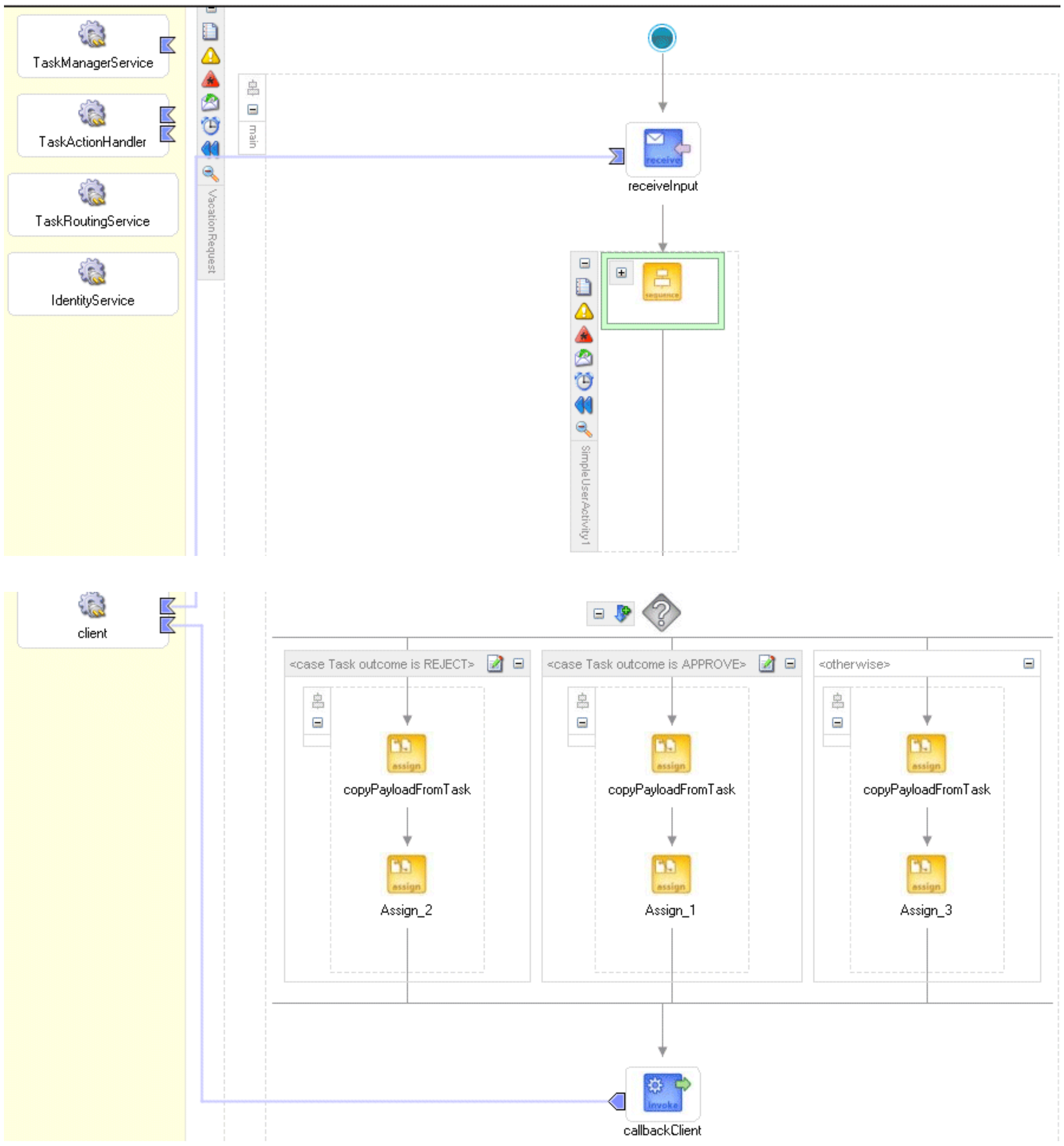


```
string ('APPROVED')
```

    c. Click **OK**.

24. For both branches in the switch—the case for reject and the case for otherwise—add an **Assign** activity with a copy rule, as you did for the approve case, but set the expression to `string('REJECTED')` instead of `string('APPROVED')`.

The business process modeling is completed. You can now deployed and test the BPEL process, which looks as follows:



Running the Example

1. Log in to Oracle BPEL Console, select the **VacationRequest** process, and go to the **Initiate** tab.
2. Enter appropriate values in each of the fields.
   - Set the creator to `jcooper`.
   - The approval task is assigned to `jstein`, who is the manager of `jcooper`.
3. Click **Post XML Message**.

4. Examine the flow of the generated instance. It is still waiting for a response from the workflow service.
5. Log in to the Worklist Application as user `jstein` and with the password `welcome`, at

   ```
   http://localhost:portnumber/integration/worklistapp/Login
   ```

   You can also select

   `Start` > `Programs` > `Oracle - Oracle_Home` > `Oracle BPEL Process Manager 10.1.2` > `Sample Worklist Application`

6. Perform any action on the task.

   You can view the task details and payload information before performing any action on the task. If the task is approved, then the business process is notified that the task has been approved. The business process determines that no additional approval is needed and marks the vacation request as approved. If the task is rejected, then the business process is notified that the task has been rejected. The business process in turn marks the vacation request as rejected. The business process is now completed.
7. Go to Oracle BPEL Console and confirm that the **VacationRequest** process has completed.

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 patterns are described, as are the components of workflow services—the task action handler, task management service, task routing service, identity service, worklist service, and notification service.

17 Worklist Application

Chapter 16, "Oracle BPEL Process Manager Workflow Services" discussed how BPEL workflow services enable you to interleave human interactions along with connectivity to systems and services into an end-to-end process flow. The BPEL worklist service provides a programmatic interface to view and manage tasks from the BPEL process. The sample Oracle BPEL Worklist Application described in this chapter is the 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 Worklist Application is layered on top of the BPEL worklist service.

This chapter contains the following topics:

- Use Cases for the Worklist Application
- Overview of Worklist Application Concepts
- Accessing the Worklist Application in Local Languages
- Customizing the Worklist Application
- Building a Worklist Application Using the Worklist Service APIs
- Building a Worklist Application Using the Worklist Service Remote APIs
- Summary

> **See Also:** Appendix F, "Demo User Community" for the organizational hierarchy of the demonstration user community used in examples throughout this chapter

Use Cases for the Worklist Application

Consider the scenario where a manager needs to approve a vacation request for an employee, or a loan agent needs to review a loan application that has been submitted as part of the BPEL process. These users typically log in to a Worklist Application to view tasks assigned to them and perform actions on these tasks. Common actions performed on tasks include updating the payload, attaching documents or comments, routing the task to other users, and completing the task by providing a conclusion such as approve or reject.

The Worklist Application is demonstrated in two samples: VacationRequest and LoanDemoPlusWithWorkflow. In the VacationRequest use case, an employee files a vacation request that is routed to his manager for approval. The manager sees the task in the worklist in the **My** tasks view. In the LoanDemoPlusWithWorkflow use case, a loan application is assigned to a LoanAgent role and then sent to two levels of approval through the management chain if the loan amount is greater than $100,000.

When any of the loan agents log in to their worklists, they see the task in their **My & Group** tasks view. One of the loan agents acquires the task and reviews it. If the loan agent approves it, the task is routed further, to two levels of management approval, if the loan amount is greater than $100,000. 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).

> **See:** `*Oracle_Home*\integration\orabpel\samples\demos` for the `VacationRequest` and `LoanDemoPlusWithWorkflow` directories

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

The Worklist Application enables users to participate in a BPEL process by performing tasks that require manual intervention. The worklist user interface displays tasks specific to the logged-in user based on the user's permissions and assigned groups and roles. In general, when a user logs in, the following types of tasks that require action are displayed:

- Tasks assigned to the user—In this case, the user has to act on the task before it is routed further.
- Tasks assigned to the groups or roles that the user belongs to—In this case, one of the users belonging to the group has to acquire the task before acting on it. If one user in a group acquires the task, it is not available to other users until it is released back to the group.

Users can review tasks for their reportees, tasks that were created by them, tasks owned by them, or any previous task that the user participated in.

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 LoanDemoPlusWithWorkflow sample or support ticket details in the HelpDeskRequest 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 as well as 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. Additionally, a user can modify flex fields, task priority, or include comments or 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.
- 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. The various system actions allowed on a task are as follows:
  - Escalate—This operation enables a user to escalate a task to his manager for further action.
  - Reassign—A manager can delegate a task to reportees. Similarly, the process owner or a user with `BPMWorkflowReassign` privileges can delegate a specific task to any other person in the organization.
  - Request More Information—Any participant in the workflow can request more information from the task creator or any of the prior assignees of the task. The user requesting more information can either have the additional information sent to him, or the user can require that the task be resubmitted through the intermediate approvers.
  - Submit More Information—This operation enables a user to respond to a request for additional information. This action is performed after the user has made the necessary updates to the task or has added comments or attachments containing additional information.
  - Route—This operation enables a user to enter an outcome and then route the task in an ad hoc fashion to the next user who must review the task.
  - Suspend—This operation enables process 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).
  - Resume—This operation enables process 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.
  - Acquire—This operation enables a user to obtain an exclusive right to work on a task that is assigned to a group or multiple users. No action can be performed on a task assigned to a group or multiple users until it is acquired. Only one user can acquire a task at any given time.
  - Release—This operation enables a user to abandon the exclusive right to work on a task that is assigned to a group or multiple users. After a task is released, any other user who is assigned to the task can acquire it.
  - 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 modeler has restricted task renewal on the workflow.
  - Withdraw—The creator of the task can withdraw a pending task. A process owner can also withdraw a task on behalf of the creator.

Logging In to the Worklist Application

Follow these instructions to access the Worklist Application.

1. Open a Web browser.

   For a list of supported browsers, see *Oracle Application Server Integration Installation Guide*.

2. Go to the following URL:

   ```
   http://hostname:9700/integration/worklistapp/Login
   ```

   where

   - `hostname` is the name of the host on which Oracle BPEL Process Manager is installed

3. Type the username and password, and click **Login**.

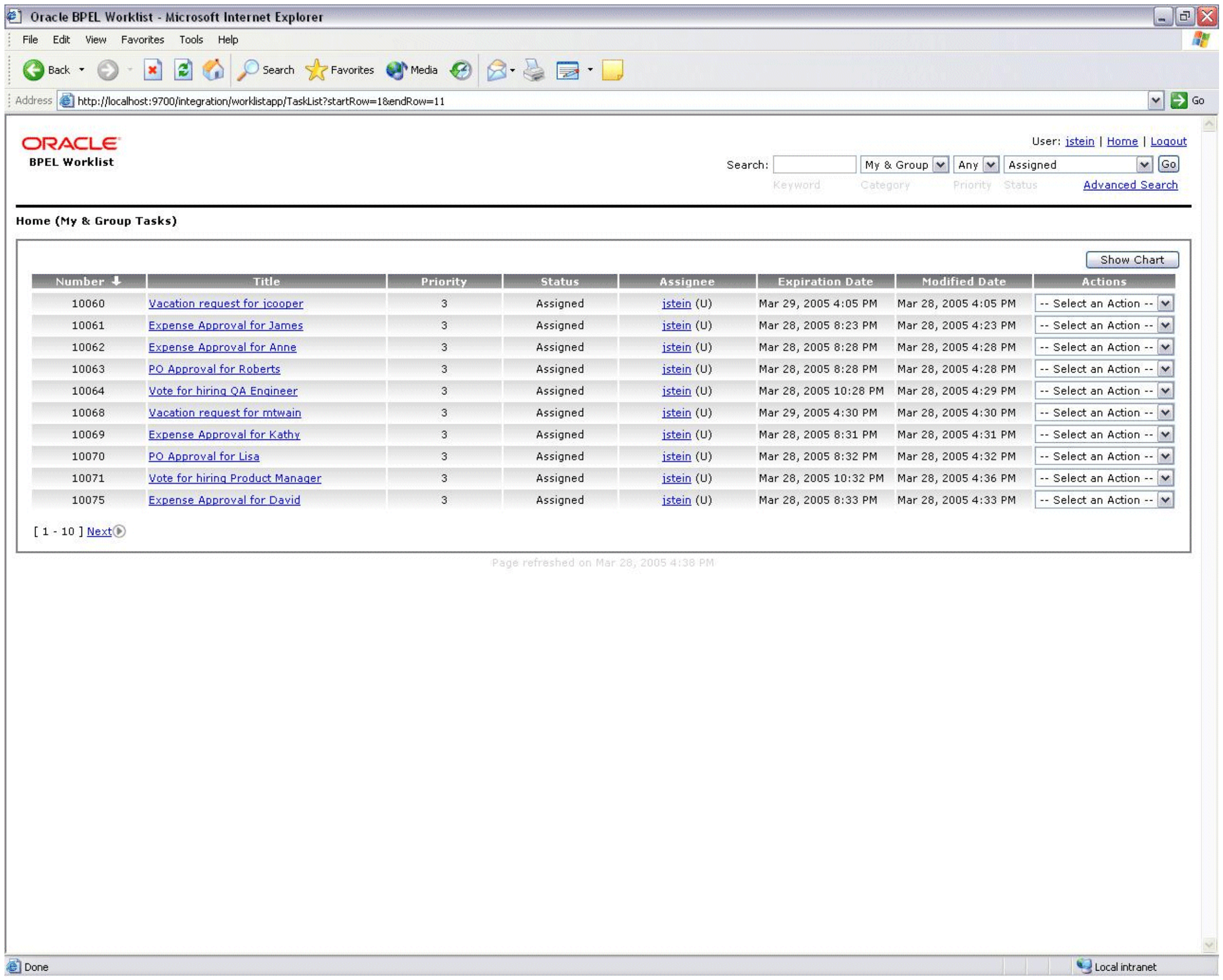
   The username and password must exist in the user community provided to JAZN. See "Configuring Identity Service" on page 16-79 for information on JAZN.

> **Note:** With some browsers, when you log in to the sample Worklist Application, the username and password fields remain blank without producing an error message. Refresh the browser so that the login information is successfully processed.

Features of the Sample Worklist Application

After you log in (see "Logging In to the Worklist Application" on page 17-4), the home page of the Worklist Application is displayed, as shown in Figure 17–1.

Figure 17–1 Worklist Application Home


Oracle BPEL Worklist - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address
Go
ORACLE
BPEL Worklist
Show Chart
Done
Local intranet

http://localhost:9700/integration/worklistapp/TaskList?startRow=1&endRow=11

User: jstein | Home | Logout

Search: [ ] My & Group | Any | Assigned | Go

Keyword Category Priority Status Advanced Search

Home (My & Group Tasks)

Number	Title	Priority	Status	Assignee	Expiration Date	Modified Date	Actions
10060	Vacation request for jcooper	3	Assigned	jstein (U)	Mar 29, 2005 4:05 PM	Mar 28, 2005 4:05 PM	-- Select an Action --
10061	Expense Approval for James	3	Assigned	jstein (U)	Mar 28, 2005 8:23 PM	Mar 28, 2005 4:23 PM	-- Select an Action --
10062	Expense Approval for Anne	3	Assigned	jstein (U)	Mar 28, 2005 8:28 PM	Mar 28, 2005 4:28 PM	-- Select an Action --
10063	PO Approval for Roberts	3	Assigned	jstein (U)	Mar 28, 2005 8:28 PM	Mar 28, 2005 4:28 PM	-- Select an Action --
10064	Vote for hiring QA Engineer	3	Assigned	jstein (U)	Mar 28, 2005 10:28 PM	Mar 28, 2005 4:29 PM	-- Select an Action --
10068	Vacation request for mtwain	3	Assigned	jstein (U)	Mar 29, 2005 4:30 PM	Mar 28, 2005 4:30 PM	-- Select an Action --
10069	Expense Approval for Kathy	3	Assigned	jstein (U)	Mar 28, 2005 8:31 PM	Mar 28, 2005 4:31 PM	-- Select an Action --
10070	PO Approval for Lisa	3	Assigned	jstein (U)	Mar 28, 2005 8:32 PM	Mar 28, 2005 4:32 PM	-- Select an Action --
10071	Vote for hiring Product Manager	3	Assigned	jstein (U)	Mar 28, 2005 10:32 PM	Mar 28, 2005 4:36 PM	-- Select an Action --
10075	Expense Approval for David	3	Assigned	jstein (U)	Mar 28, 2005 8:33 PM	Mar 28, 2005 4:33 PM	-- Select an Action --

[1 - 10] Next

Page refreshed on Mar 28, 2005 4:38 PM

From this page, you can retrieve worklist tasks by using the **Search** field to do a keyword search or by using the **Task Filter**, **Priority**, and **Status** fields to specify search criteria. Table 17–1 describes the salient features of the Worklist Application home page shown in Figure 17–1.

Table 17–1 Contents of the Worklist Application Home

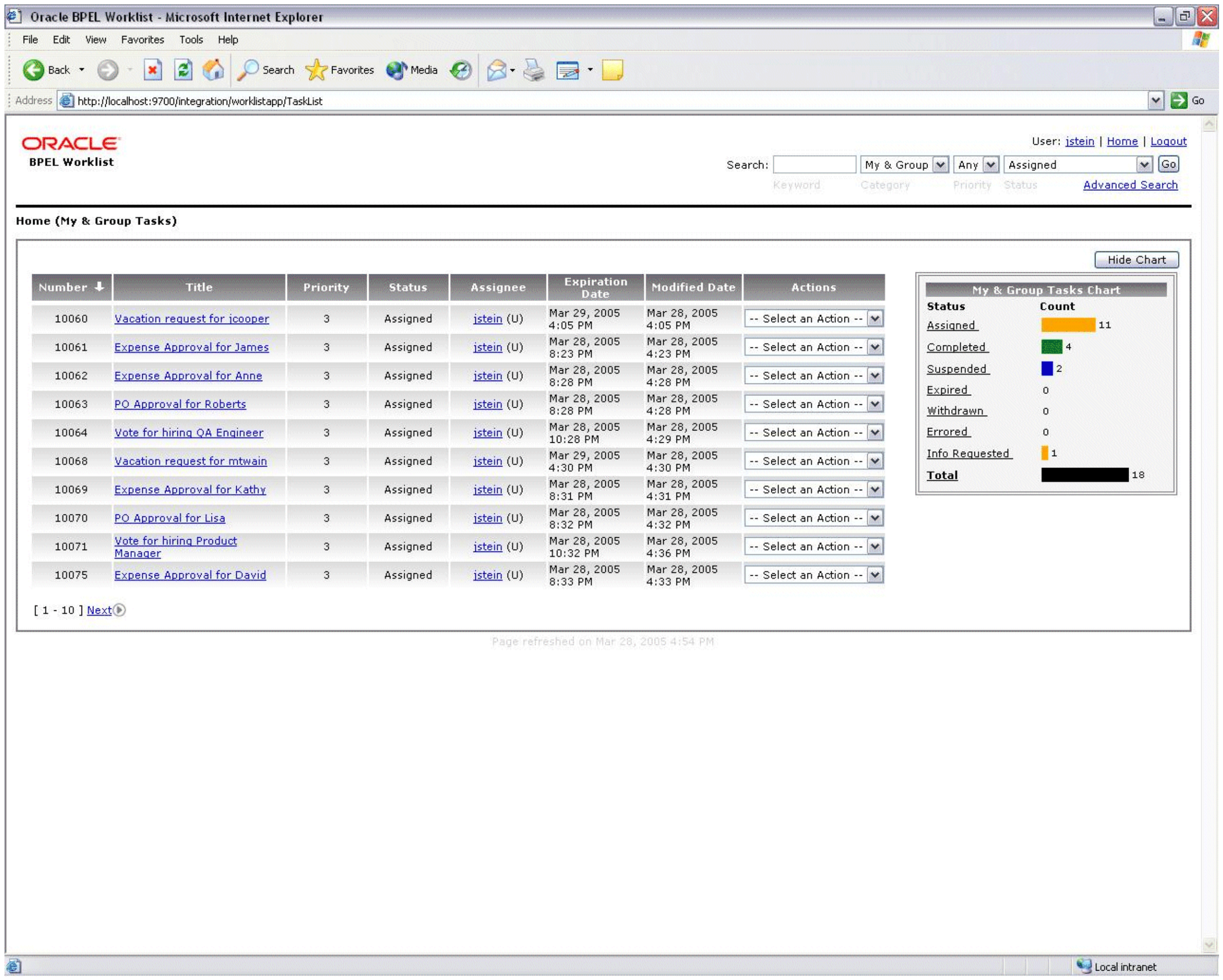
Page Element	Location in Page	Description
User link	top right of Figure 17–1	Click the user name link to see the logged-in user's full name, contact information, title, manager's name, reportees' names, and assigned groups and roles. The manager and reportee names are also links to the same information on those individuals.
Search Keyword field	top center of Figure 17–1	Enter a keyword to search task titles, comments, identification keys, and the flex string fields of tasks that qualify for the specified filter criterion.
Task Filter list	top right of Figure 17–1	Select from the following: ■ **My**—retrieves tasks directly assigned to the logged-in user ■ **Group**—tasks assigned to the groups to which the logged-in user belongs ■ **My & Group**—tasks assigned to the user and the groups to which the logged-in user belongs ■ **Reportees**—tasks assigned to the users who report to the logged-in user ■ **Owner**—tasks that are owned by the logged-in user by way of process ownership ■ **Creator**—tasks that were created or initiated by the logged-in user ■ **Previous**—tasks that the logged-in user has updated ■ **Admin**—appears only if the logged-in user has been granted the BPMWorkflowAdmin role ■ **All**—tasks from all of the above
Priority list	top right of Figure 17–1	Select from **Any** or **1** through **5**, where **1** is the highest priority.
Status list	top right of Figure 17–1	Select from the following: ■ **Any** ■ **Assigned** ■ **Completed** ■ **Suspended** (can be resumed later) ■ **Withdrawn** ■ **Expired** ■ **Errored** (errored while processing; a user with the `BPMWorkflowAdmin` role can troubleshoot the problem.) ■ **Information Requested**
Go button	top right of Figure 17–1	Click **Go** after selecting from the search fields.
Advanced Search link		Click to go to the Advanced Search page, which provides additional search filters.
Show (Hide) Chart button	right side of Figure 17–1	Shows a bar chart of the listed tasks in the selected task filter, broken down by status. See Figure 17–2 for an example.
Number column	right side of Figure 17–1	Displays the task number for the task.
Title column	middle of Figure 17–1	Tasks associated with a purged or archived process instance do not appear. Click the column name for ascending or descending sorts.
Priority column	middle of Figure 17–1	Click the column name for ascending or descending sorts.

Table 17–1 (Cont.) Contents of the Worklist Application Home

Page Element	Location in Page	Description
Status column	middle of Figure 17–1	Status states are as follows: **Assigned**, **Completed**, **Errored**, **Expired**, **Info Requested**, **Stale**, **Suspended**, and **Withdrawn**. Click the column name for ascending or descending sorts.
Assignees column	middle of Figure 17–1	Person or group to whom the task is assigned.
Expiration Date column	middle of Figure 17–1	Date and time the tasks expires. Click the column name for ascending or descending sorts.
Modified Date column	middle of Figure 17–1	Date and time the task was modified. Click the column name for ascending or descending sorts.
Actions column	middle of Figure 17–1	The possible actions that apply to the task. See"Task Actions" on page 17-10 for more information.
Next or **Previous** links	bottom left of Figure 17–1	Click **Previous** or **Next** to go the previous or next page in the list. The absolute row numbers of the tasks are shown in square brackets. The size of the page (number of rows per page) is controlled by an optional parameter, `oracle.tip.worklist.sampes.tasklist.maxrows`, in the file `pc.properties`, which appears in *`Oracle_Home`*`\integration\orabpel\system\services\config`. The default value is `20`.

If you click **Show Chart**, a bar chart of the tasks is displayed, as shown in Figure 17–2.

Figure 17–2 Worklist Application Home with Chart Displayed


Oracle BPEL Worklist - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:9700/integration/worklistapp/TaskList
Go
ORACLE
BPEL Worklist
User: jstein | Home | Logout
Search:
My & Group
Any
Assigned
Go
Keyword Category Priority Status
Advanced Search
Hide Chart
Local intranet

Home (My & Group Tasks)

Number	Title	Priority	Status	Assignee	Expiration Date	Modified Date	Actions
10060	Vacation request for jcooper	3	Assigned	jstein (U)	Mar 29, 2005 4:05 PM	Mar 28, 2005 4:05 PM	-- Select an Action --
10061	Expense Approval for James	3	Assigned	jstein (U)	Mar 28, 2005 8:23 PM	Mar 28, 2005 4:23 PM	-- Select an Action --
10062	Expense Approval for Anne	3	Assigned	jstein (U)	Mar 28, 2005 8:28 PM	Mar 28, 2005 4:28 PM	-- Select an Action --
10063	PO Approval for Roberts	3	Assigned	jstein (U)	Mar 28, 2005 8:28 PM	Mar 28, 2005 4:28 PM	-- Select an Action --
10064	Vote for hiring QA Engineer	3	Assigned	jstein (U)	Mar 28, 2005 10:28 PM	Mar 28, 2005 4:29 PM	-- Select an Action --
10068	Vacation request for mtwain	3	Assigned	jstein (U)	Mar 29, 2005 4:30 PM	Mar 28, 2005 4:30 PM	-- Select an Action --
10069	Expense Approval for Kathy	3	Assigned	jstein (U)	Mar 28, 2005 8:31 PM	Mar 28, 2005 4:31 PM	-- Select an Action --
10070	PO Approval for Lisa	3	Assigned	jstein (U)	Mar 28, 2005 8:32 PM	Mar 28, 2005 4:32 PM	-- Select an Action --
10071	Vote for hiring Product Manager	3	Assigned	jstein (U)	Mar 28, 2005 10:32 PM	Mar 28, 2005 4:36 PM	-- Select an Action --
10075	Expense Approval for David	3	Assigned	jstein (U)	Mar 28, 2005 8:33 PM	Mar 28, 2005 4:33 PM	-- Select an Action --

[1 - 10] Next

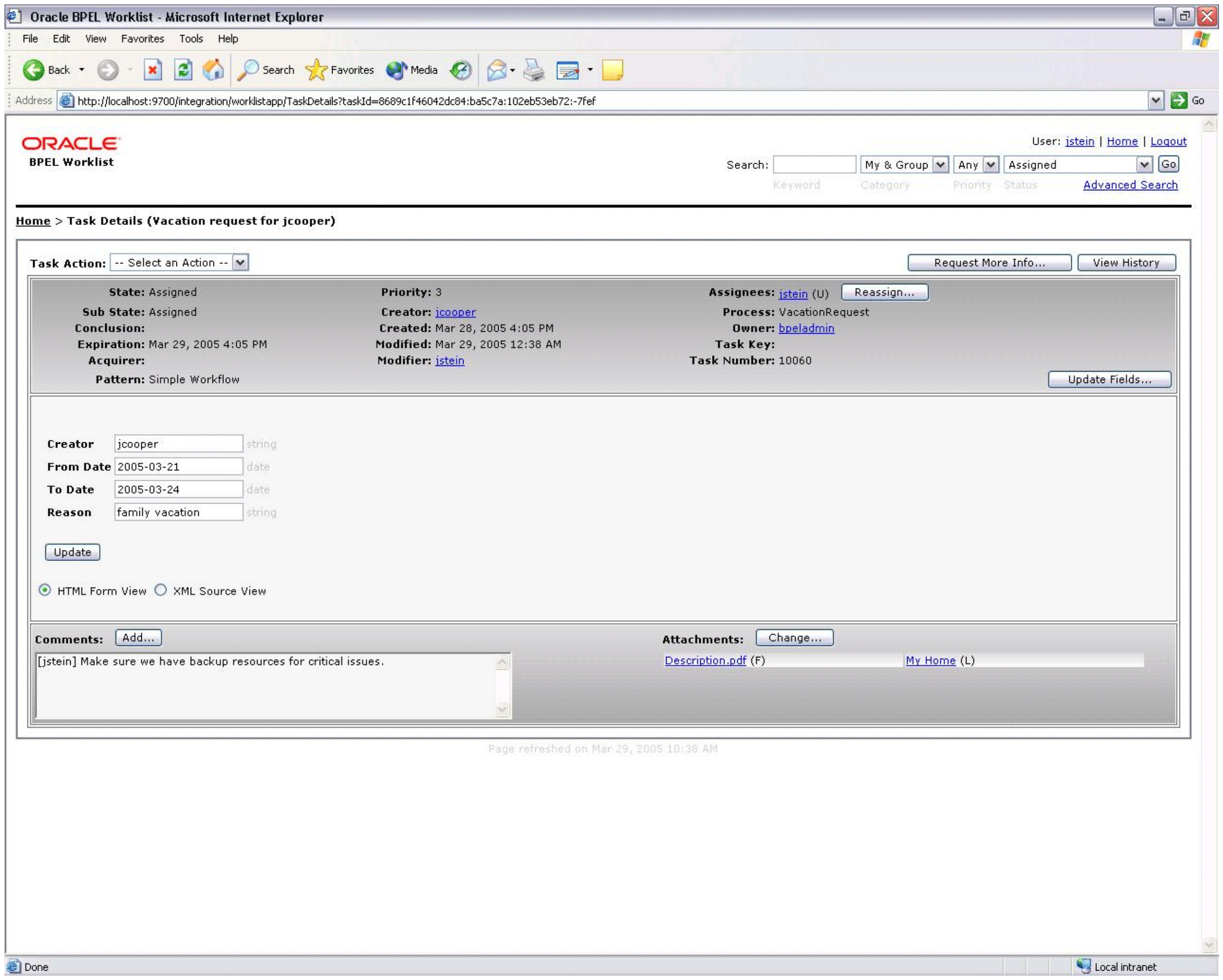
My & Group Tasks Chart

Status	Count
Assigned	11
Completed	4
Suspended	2
Expired	0
Withdrawn	0
Errored	0
Info Requested	1
Total	18

Page refreshed on Mar 28, 2005 4:54 PM

If you click a task in the **Title** column, the Task Details page is displayed, as shown in Figure 17–3.

Figure 17–3 Task Details Page


Oracle BPEL Worklist - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:9700/integration/worklistapp/TaskDetails?taskId=8689c1f46042dc84:ba5c7a:102eb53eb72:-7fef
Go
ORACLE
BPEL Worklist
User: jstein | Home | Logout
Search: My & Group Any Assigned Go
Keyword Category Priority Status Advanced Search

Home > Task Details (Vacation request for jcooper)

Task Action: -- Select an Action --   Request More Info...   View History

State: Assigned
Sub State: Assigned
Conclusion:
Expiration: Mar 29, 2005 4:05 PM
Acquirer:
Pattern: Simple Workflow
Priority: 3
Creator: jcooper
Created: Mar 28, 2005 4:05 PM
Modified: Mar 29, 2005 12:38 AM
Modifier: jstein
Assignees: jstein (U)   Reassign...
Process: VacationRequest
Owner: bpeladmin
Task Key:
Task Number: 10060
Update Fields...

Creator	jcooper	string
From Date	2005-03-21	date
To Date	2005-03-24	date
Reason	family vacation	string

Update

HTML Form View   XML Source View

Comments: Add...
[jstein] Make sure we have backup resources for critical issues.
Attachments: Change...
Description.pdf (F)   My Home (L)

Page refreshed on Mar 29, 2005 10:38 AM

Done   Local intranet

Table 17–2 describes the Task Details page shown in Figure 17–3, and also references other figures in this chapter. (The table does not include page elements, such as the **User** link or the search fields, already described in Table 17–1.)

Table 17–2 Contents of the Task Details Page

Page Element	Location in Page	Description
Task Action list	top left of Figure 17–3	Shows the actions you can perform on a task, such as approving a vacation request or escalating a purchasing decision, depending on how the process flow was defined in JDeveloper BPEL Designer. See "Task Actions" on page 17-10 for more information.
Route... button	top right of Figure 17–4	Routes the task to another user. The button appears only if the task was designed to support dynamic routing (that is, no predetermined set of approvers, for example). See "Routing" on page 17-12 for more information.
Request More Info... button	top right of Figure 17–3	Requests more information from the task creator or a previous approver in a simple or sequential approval process. See "Requesting More Information" on page 17-13 for more information.
View SubTasks button	top right of Figure 17–16	Click to go to the Subtasks page. This button is available only for a parent task of a parallel flow. See "Parallel Tasks" on page 17-23 for more information.
View History button	top right of Figure 17–16	Click to see the complete trail of changes made to a task. See "Task History and Sequence (Version) Numbers" on page 17-14 for more information.

Table 17–2 (Cont.) Contents of the Task Details Page

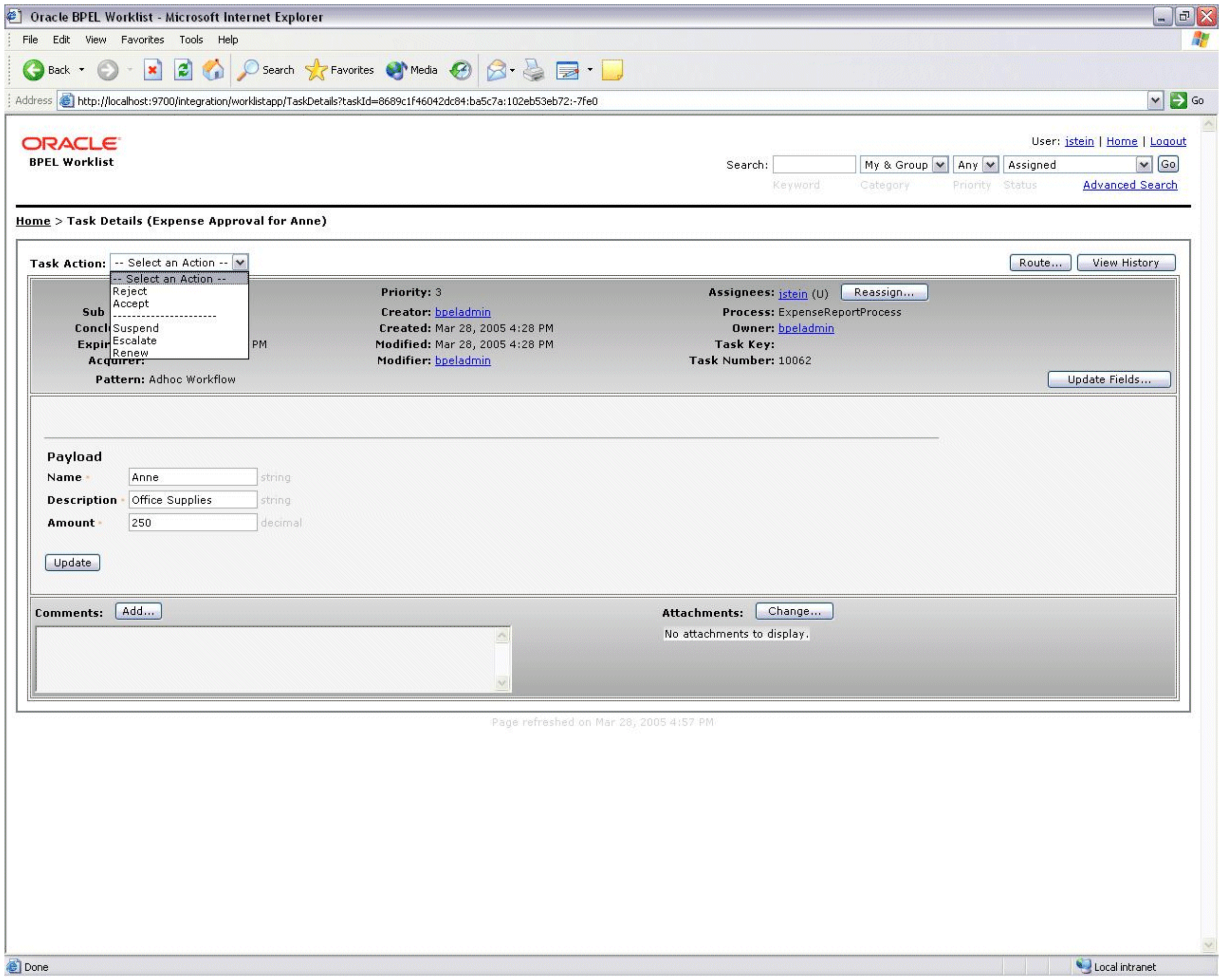
Page Element	Location in Page	Description
Header section	top of Figure 17–3	The header includes the process name, state, and priority, and information about who created, modified, acquired, or is assigned to the task. It also displays dates related to task creation, last modification, and expiration. The task flow pattern is also displayed.
Payload section	middle of Figure 17–3	Click **HTML Form View** (default) or **XML Source View** for the display options. See "The Payload" on page 17-17 for more information.
Comments and **Attachments** section	bottom of Figure 17–3	Click the **Add** button in the **Comments** section to add comments. Click the **Change...** button in the **Attachments** area to add or change an attachment. See "Comments" on page 17-19 and "Attachments" on page 17-20 for more information.
Reassign button	lower left of Figure 17–15	Click this button if you want to change the assignment of a task to other users. You must have permission to reassign a task, which is granted through the `BPMWorkflowReassign` role. See "Reassignment" on page 17-22 for more information.
Update Fields button	middle right of Figure 17–16	Click to go to the edit page for updating task header fields such as priority, identification key, and flex fields.

The following sections describe activities provided by the Worklist Application.

Task Actions

Figure 17–4 shows the **Task Action** list. The tasks 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, such as **Accept** or **Reject**, are listed first. Custom actions are defined in JDeveloper BPEL Designer when you define the business process. System actions, such as **Escalate** or **Suspend**, are listed below a separator line.

Figure 17–4 Task Actions



Oracle BPEL Worklist - Microsoft Internet Explorer

Address http://localhost:9700/integration/worklistapp/TaskDetails?taskId=8689c1f46042dc84:ba5c7a:102eb53eb72:-7fe0

BPEL Worklist

User: jstein | Home | Logout

Search: My & Group | Any | Assigned | Go

Keyword Category Priority Status Advanced Search

Home > Task Details (Expense Approval for Anne)

Task Action: -- Select an Action --

-- Select an Action --
Reject
Accept

Suspend
Escalate
Renew

Route... View History

Sub
Concl
Expir PM
Acquirer:
Pattern: Adhoc Workflow

Priority: 3
Creator: bpeladmin
Created: Mar 28, 2005 4:28 PM
Modified: Mar 28, 2005 4:28 PM
Modifier: bpeladmin

Assignees: jstein (U) Reassign...
Process: ExpenseReportProcess
Owner: bpeladmin
Task Key:
Task Number: 10062

Update Fields...

Payload

Name	Anne	string
Description	Office Supplies	string
Amount	250	decimal

Update

Comments: Add...

Attachments: Change...

No attachments to display.

Page refreshed on Mar 28, 2005 4:57 PM

Done Local intranet

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.

System actions are described in Table 17–3.

Table 17–3 System Task Actions

Action	Description
Acquire	If a task is assigned to a group or multiple users, then the task must be acquired first. **Acquire** is the only action available in the **Task Action** list. After a task is acquired, all applicable actions are listed.
Escalate	If you are not able to complete a task, you can escalate it and add an optional comment in the **Comments** area. The task is reassigned to your manager.
Release	If a task is assigned to a group or multiple users, it can be released if the user who acquired the task cannot complete the task. Any of the other assignees can acquire and complete the task.

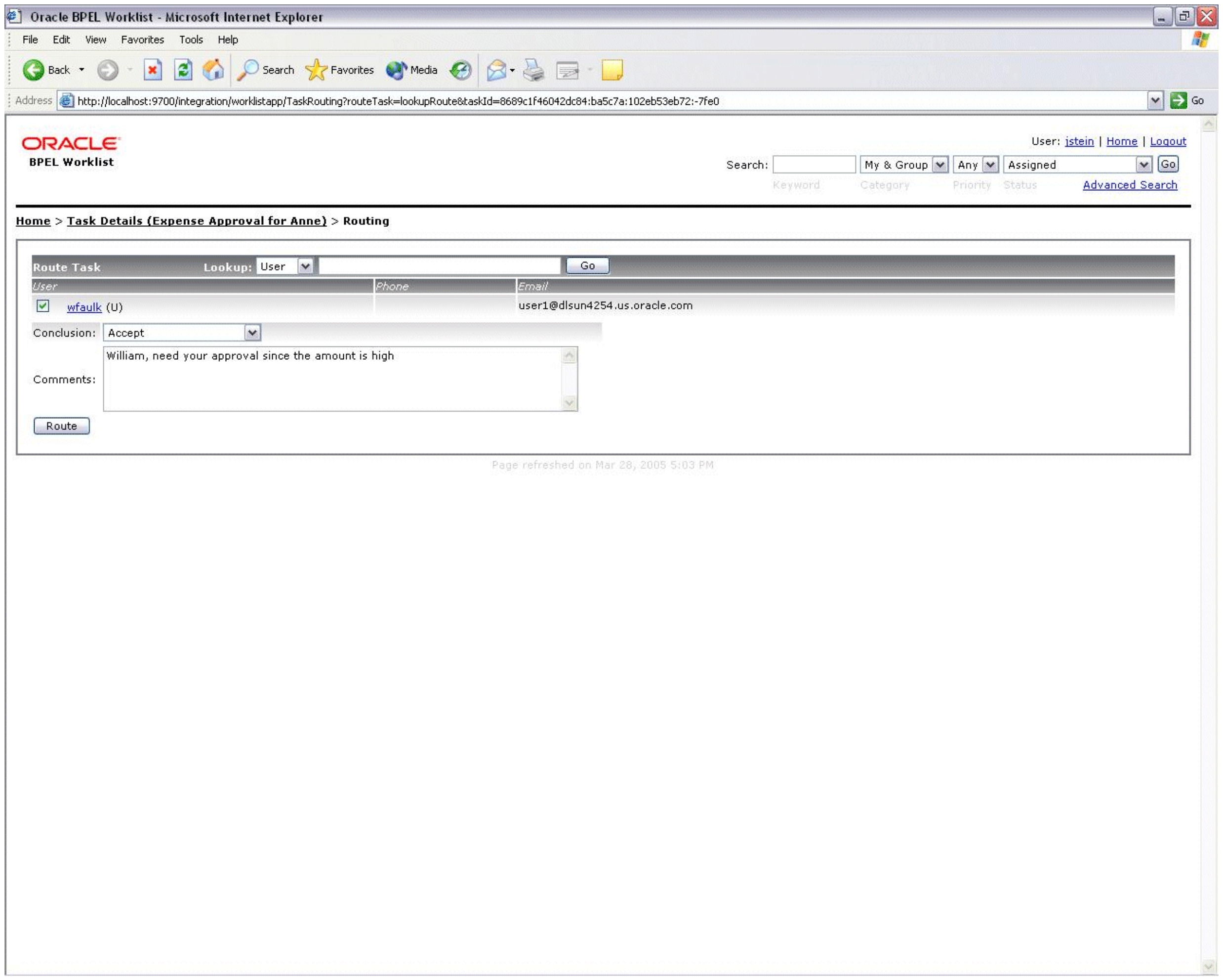
Table 17–3 (Cont.) System Task Actions

Action	Description
Renew	If a task is about to expire, you can renew it and add an optional comment in the **Comments** area. The task expiration date is extended one week. A renewal appears in the task history. The renewal duration for a task can be controlled by an optional parameter, `oracle.tip.worklist.samples.taskactin.renew.duration`, in the file `pc.properties`, which appears in *`Oracle_Home`*`\integration\orabpel\system\services\config`. The default value is `P7D` (seven days).
Submit More Info	Use this action if another user requests that you supply more information. If reapproval is not required, then the task is assigned to the next approver or the next step in the business process.
Suspend and **Resume**	If a task is not relevant at present, you can suspend it. These options are available only to users who have been granted the BPMWorkflowSuspend role. Other users can access the task by selecting **Previous** in the task filter or by looking up tasks in the **Suspended** status. Buttons that update a task are disabled after suspension.
Withdraw	If you are the creator of a task and do not want to continue with it, for example, you want to cancel a vacation request, you can withdraw it and add an optional comment in the **Comments** area. The business process determines what happens next. You can use the **Withdraw** action on the home page by using the **Creator** task filter.

Routing

If there is no predetermined sequence of approvers and the actual set of approvers is determined dynamically, then the task can be routed in an ad hoc fashion. Note that the process must have been designed for supporting ad hoc routing. For such tasks, a **Route** button appears on the Task Details page, as shown in Figure 17–5.

Figure 17–5 Routing a Task


Oracle BPEL Worklist - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address
ORACLE
BPEL Worklist
Keyword Category Priority Status
Advanced Search

http://localhost:9700/integration/worklistapp/TaskRouting?routeTask=lookupRoute&taskId=8689c1f46042dc84:ba5c7a:102eb53eb72:-7fe0

User: jstein | Home | Logout

Search: My & Group Any Assigned Go

Home > Task Details (Expense Approval for Anne) > Routing

Route Task Lookup: User Go

User	Phone	Email
✓ wfaulk (U)		user1@dlsun4254.us.oracle.com

Conclusion: Accept

Comments: William, need your approval since the amount is high

Route

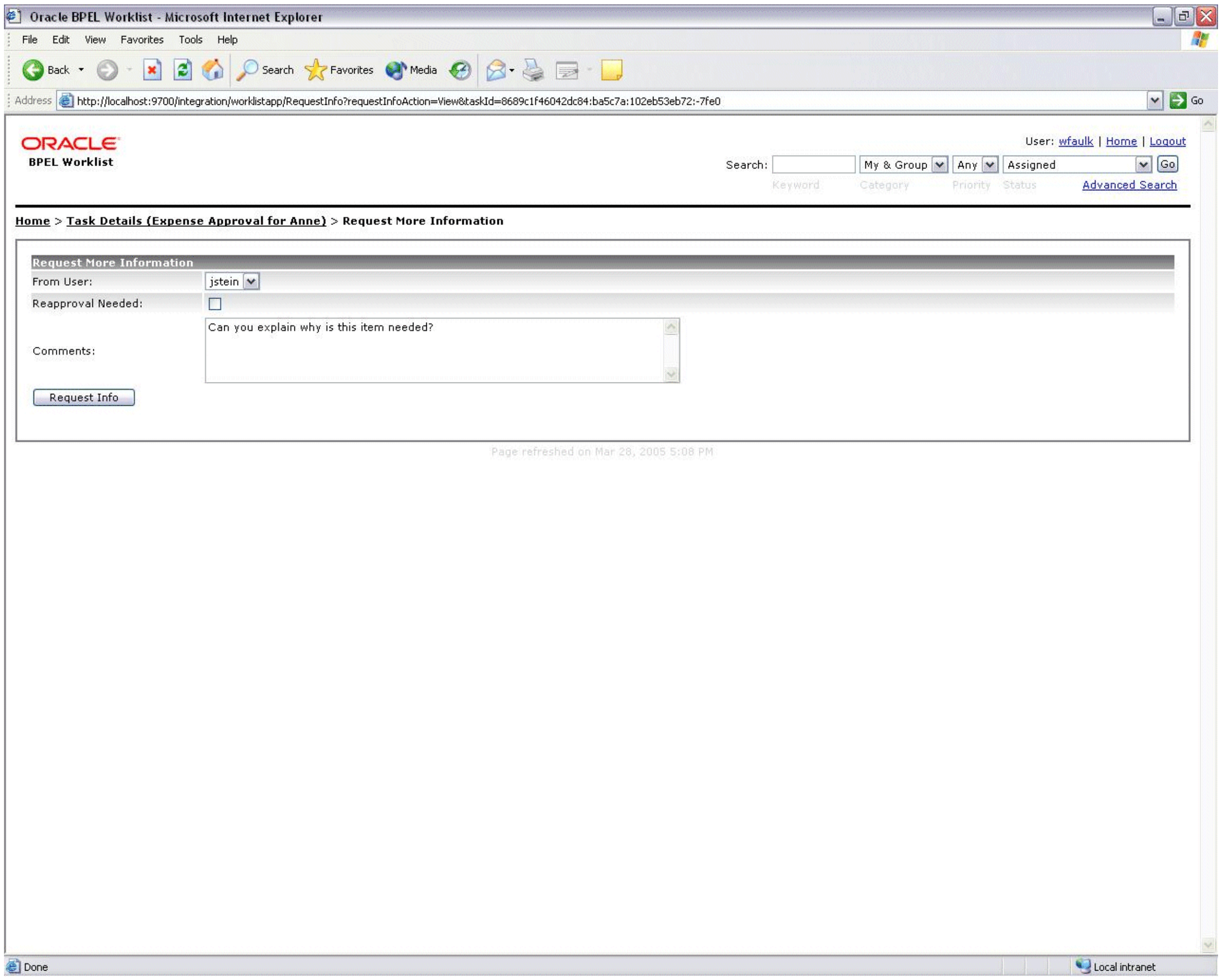
Page refreshed on Mar 28, 2005 5:03 PM

Clicking **Route** displays the ad hoc Routing page. Like the Reassign page, the user can look up the next approver by selecting **My & Group** and providing a complete or wildcard search string name. The difference between reassign and route is that in the case of route, the user's conclusion is added to the task and the task is sent to the new user as the next approver in the ad hoc sequence. Note that the new user or users are selected for the next phase of approval only. A chain or sequence of approvals is not denoted. The user must select a conclusion and can optionally add comments before routing the task.

Requesting More Information

Figure 17–6 shows where you request more information. 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, as described in Table 17–3.

Figure 17–6 Requesting More Information


Oracle BPEL Worklist - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:9700/integration/worklistapp/RequestInfo?requestInfoAction=View&taskId=8689c1f46042dc84:ba5c7a:102eb53eb72:-7fe0
Go
ORACLE
BPEL Worklist
User: wfaulk | Home | Logout
Search: My & Group Any Assigned Go
Keyword Category Priority Status Advanced Search
Done
Local intranet

Home > Task Details (Expense Approval for Anne) > Request More Information

Request More Information

From User:	jstein
Reapproval Needed:	☐
Comments:	Can you explain why is this item needed?

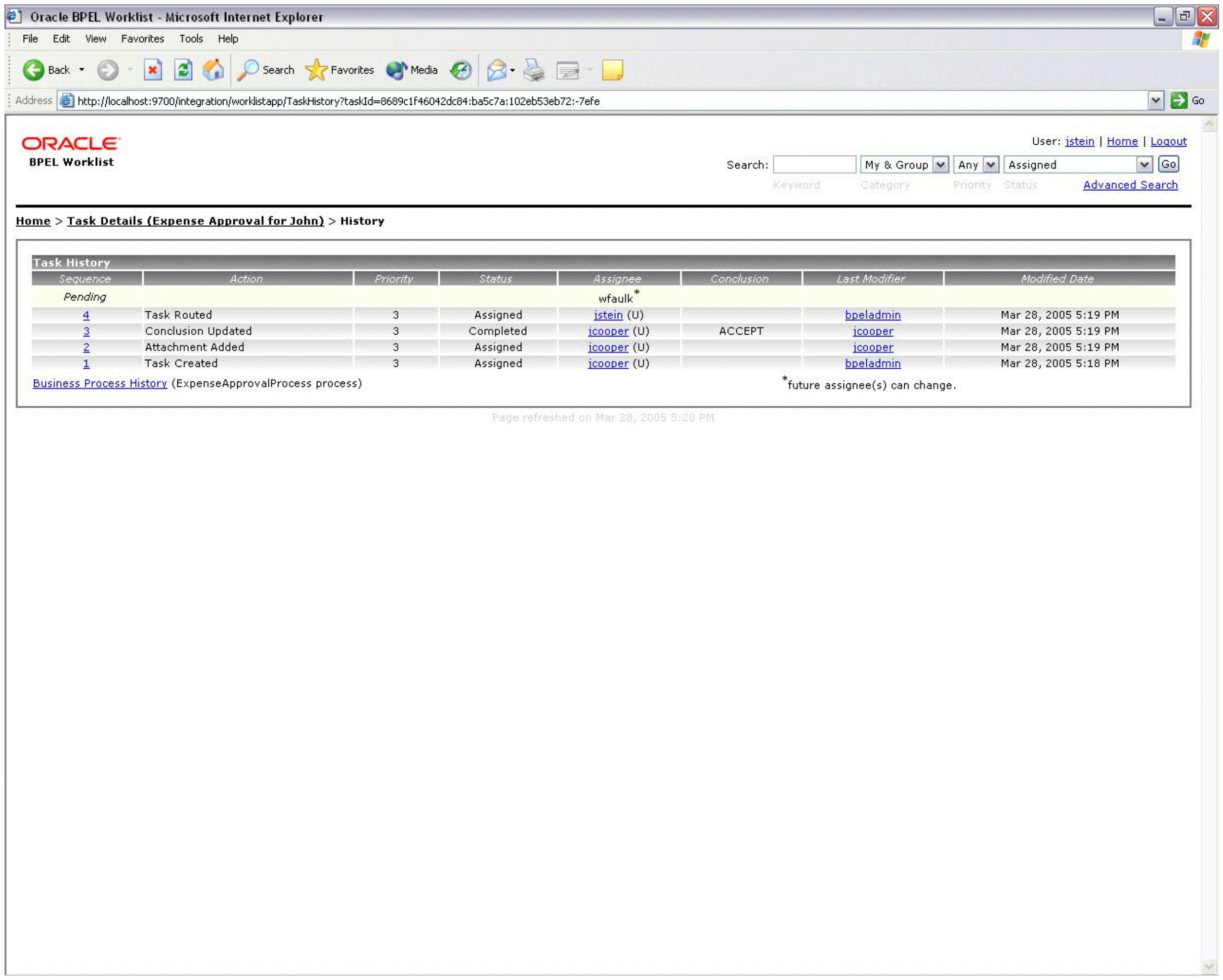
Request Info

Page refreshed on Mar 28, 2005 5:08 PM

Task History and Sequence (Version) Numbers

Figure 17–7 and Figure 17–8 show task history and sequence numbers (versions). Each time a task is updated or an action is performed, and sequencing (versioning) has been requested, a new sequence number is created. Task sequence numbers are automatically created for important actions and changes, such as adding and deleting attachments and payloads.

Figure 17–7 Task HIstory


Oracle BPEL Worklist - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:9700/integration/worklistapp/TaskHistory?taskId=8689c1f46042dc84:ba5c7a:102eb53eb72:-7efe
Go
ORACLE
BPEL Worklist
User: jstein | Home | Logout
Search: My & Group Any Assigned Go
Keyword Category Priority Status Advanced Search

Home > Task Details (Expense Approval for John) > History

Task History

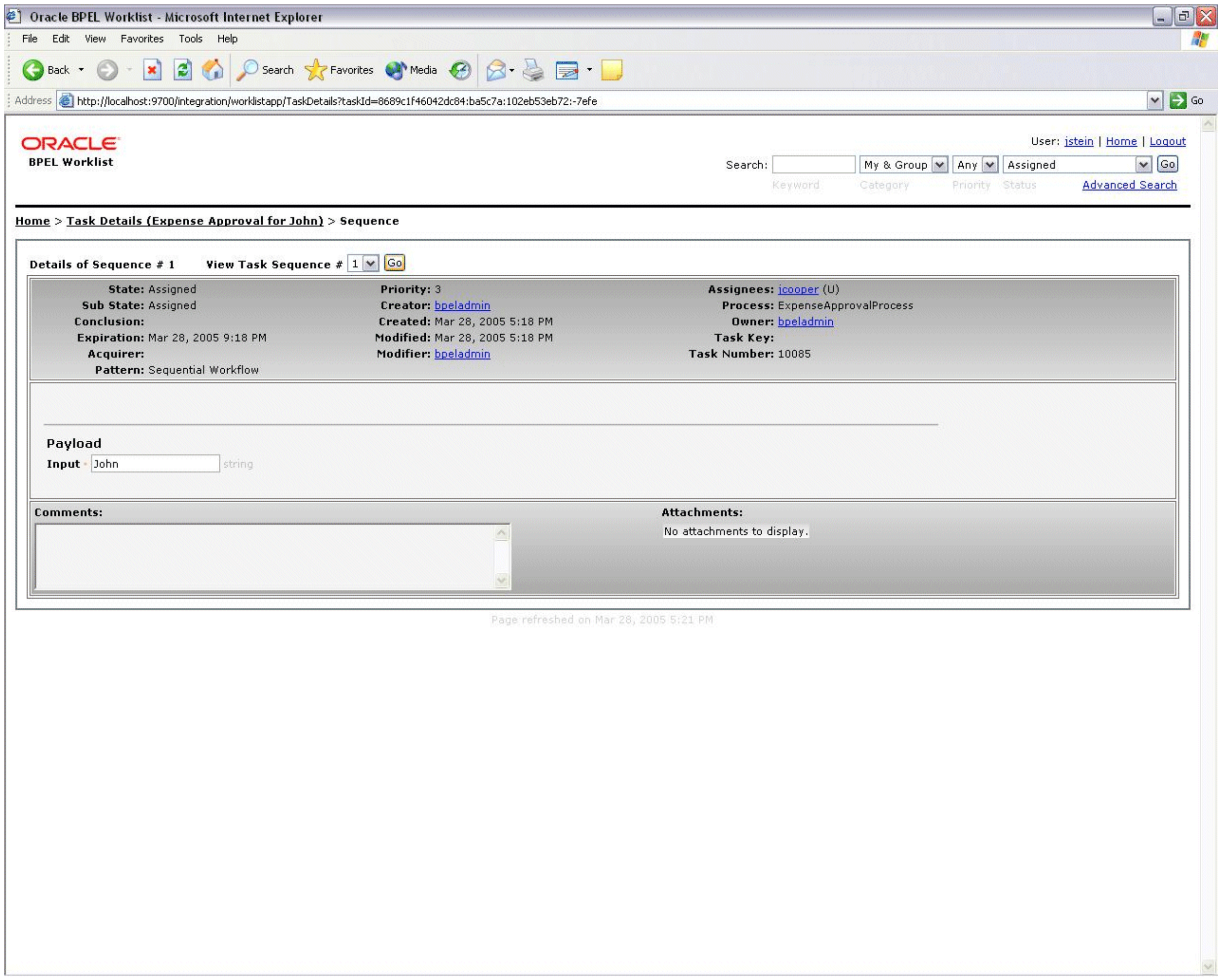
Sequence	Action	Priority	Status	Assignee	Conclusion	Last Modifier	Modified Date
Pending				wfaulk*			
4	Task Routed	3	Assigned	jstein (U)		bpeladmin	Mar 28, 2005 5:19 PM
3	Conclusion Updated	3	Completed	jcooper (U)	ACCEPT	jcooper	Mar 28, 2005 5:19 PM
2	Attachment Added	3	Assigned	jcooper (U)		jcooper	Mar 28, 2005 5:19 PM
1	Task Created	3	Assigned	jcooper (U)		bpeladmin	Mar 28, 2005 5:18 PM

Business Process History (ExpenseApprovalProcess process)

*future assignee(s) can change.

Page refreshed on Mar 28, 2005 5:20 PM

Figure 17–8 Task Sequence (Version) Numbers


Oracle BPEL Worklist - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:9700/integration/worklistapp/TaskDetails?taskId=8689c1f46042dc84:ba5c7a:102eb53eb72:-7efe
Go
ORACLE
BPEL Worklist
User: jstein | Home | Logout
Search: My & Group Any Assigned Go
Keyword Category Priority Status Advanced Search
Home > Task Details (Expense Approval for John) > Sequence
Details of Sequence # 1 View Task Sequence # 1 Go

State: Assigned	**Priority:** 3	**Assignees:** jcooper (U)
Sub State: Assigned	**Creator:** bpeladmin	**Process:** ExpenseApprovalProcess
Conclusion:	**Created:** Mar 28, 2005 5:18 PM	**Owner:** bpeladmin
Expiration: Mar 28, 2005 9:18 PM	**Modified:** Mar 28, 2005 5:18 PM	**Task Key:**
Acquirer:	**Modifier:** bpeladmin	**Task Number:** 10085
Pattern: Sequential Workflow		

Payload
Input · John string

Comments:

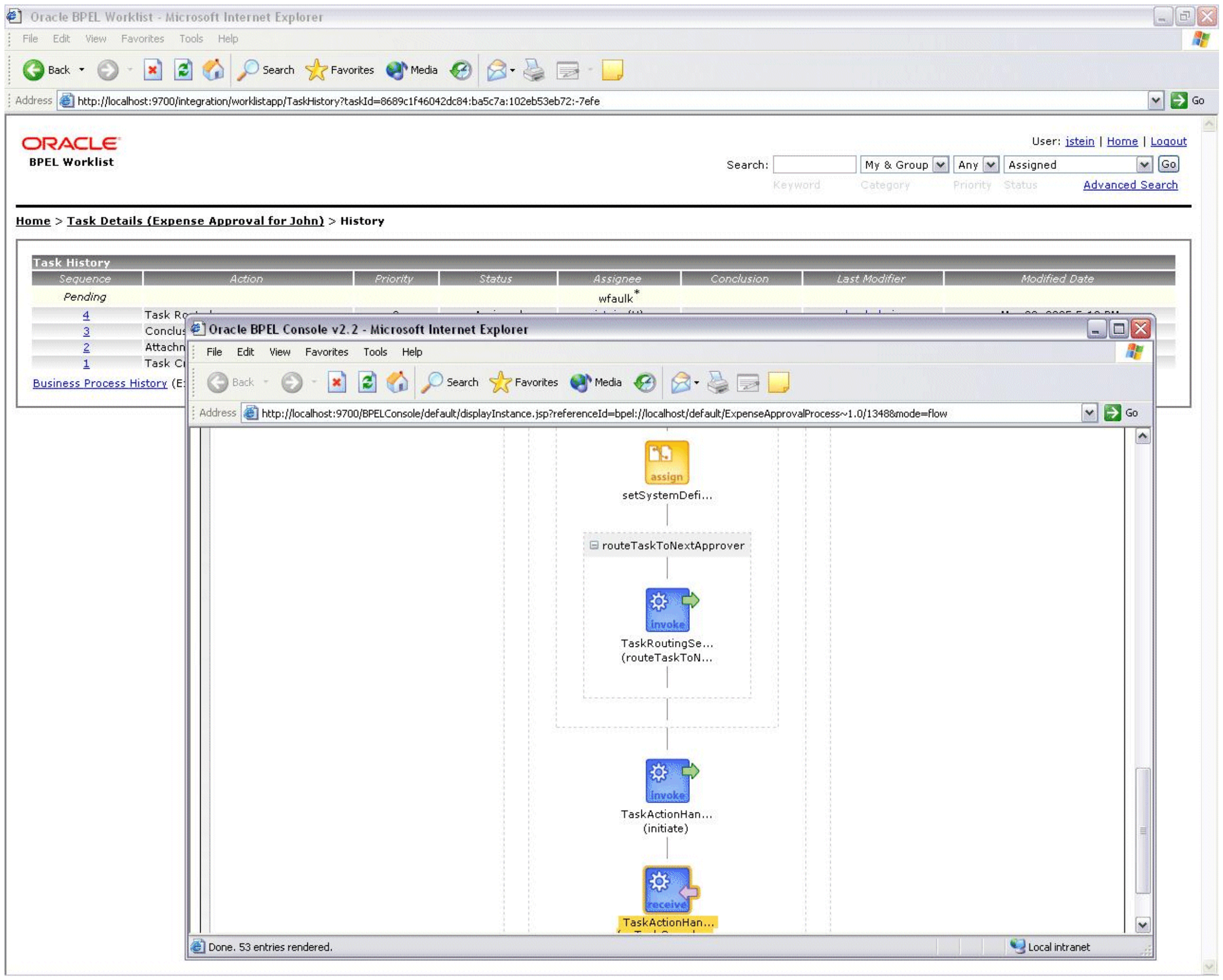
Attachments:
No attachments to display.

Page refreshed on Mar 28, 2005 5:21 PM

If a task requires sequential approval, the list of approvers is displayed.

You can also view the flow of the corresponding business process by clicking the **Business Process History** link at the bottom of the page. Oracle BPEL Console containing the process flow is displayed, as shown in Figure 17–9.

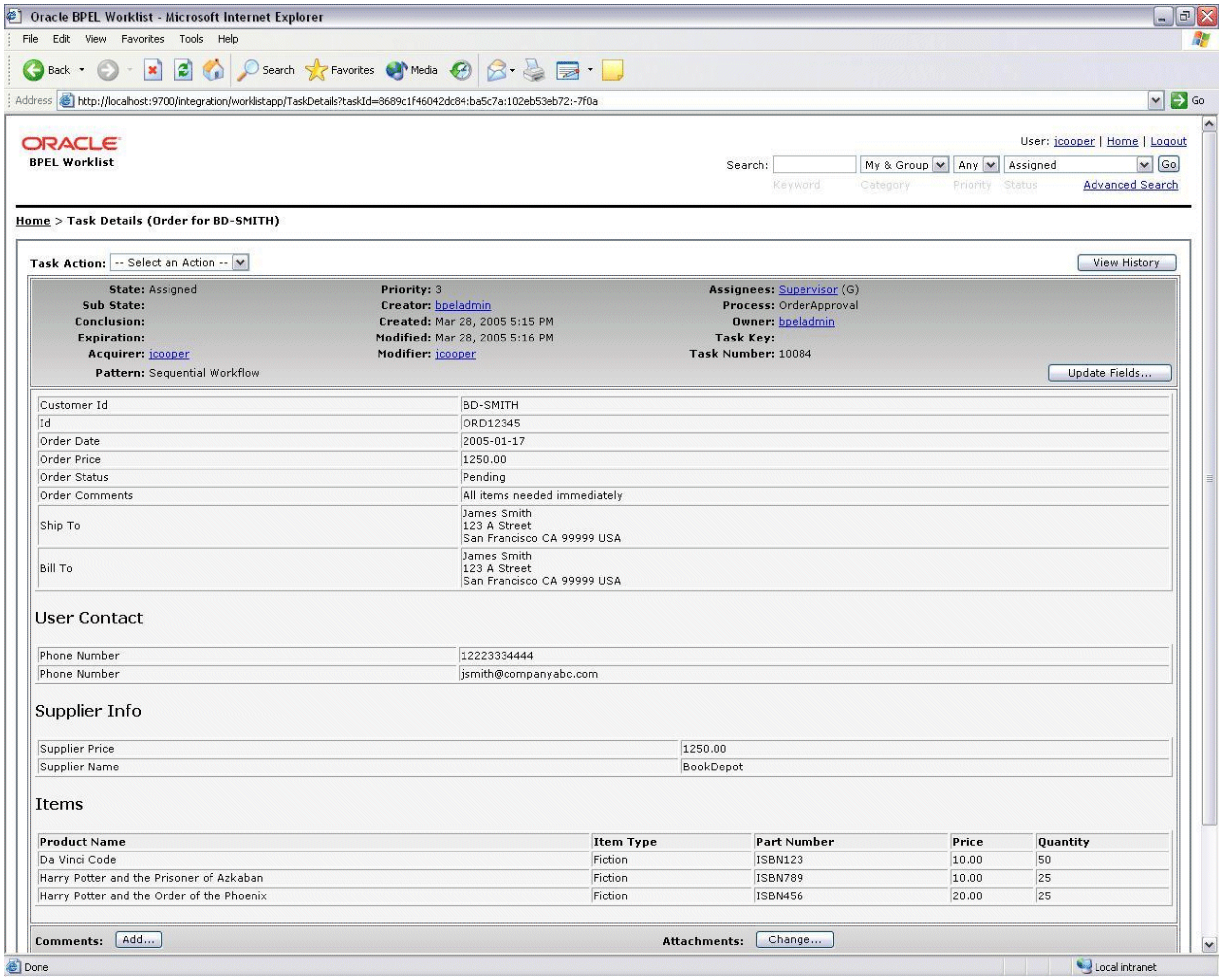
Figure 17–9 Process Flow


Oracle BPEL Worklist - Microsoft Internet Explorer
http://localhost:9700/integration/worklistapp/TaskHistory?taskId=8689c1f46042dc84:ba5c7a:102eb53eb72:-7efe
ORACLE
BPEL Worklist
User: jstein | Home | Logout
Search:
My & Group
Any
Assigned
Go
Keyword
Category
Priority
Status
Advanced Search
Home > Task Details (Expense Approval for John) > History
Task History
Sequence
Action
Priority
Status
Assignee
Conclusion
Last Modifier
Modified Date
Pending
wfaulk
Business Process History
Oracle BPEL Console v2.2 - Microsoft Internet Explorer
http://localhost:9700/BPELConsole/default/displayInstance.jsp?referenceId=bpel://localhost/default/ExpenseApprovalProcess~1.0/1348&mode=flow
assign
setSystemDefi...
routeTaskToNextApprover
invoke
TaskRoutingSe...
(routeTaskToN...
invoke
TaskActionHan...
(initiate)
receive
TaskActionHan...
Done. 53 entries rendered.
Local intranet

The Payload

Figure 17–10 shows a payload in the HTML form view using a customized XSL template to render the original XML payload in an HTML format. The HTML form view can also be displayed using a custom payload JSP page. See Chapter 16, "Oracle BPEL Process Manager Workflow Services" for information on using an XSL template or using the autogenerated JSP.

Figure 17–10 Task Payload (XSL Transformed Payload)


Oracle BPEL Worklist - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Search
Favorites
Media
Address
http://localhost:9700/integration/worklistapp/TaskDetails?taskId=8689c1f46042dc84:ba5c7a:102eb53eb72:-7f0a
Go
ORACLE
BPEL Worklist
User: jcooper | Home | Logout
Search:
My & Group
Any
Assigned
Go
Keyword
Category
Priority
Status
Advanced Search
Done
Local intranet

Home > Task Details (Order for BD-SMITH)

Task Action: -- Select an Action --   View History

State: Assigned	**Priority:** 3	**Assignees:** Supervisor (G)
Sub State:	**Creator:** bpeladmin	**Process:** OrderApproval
Conclusion:	**Created:** Mar 28, 2005 5:15 PM	**Owner:** bpeladmin
Expiration:	**Modified:** Mar 28, 2005 5:16 PM	**Task Key:**
Acquirer: jcooper	**Modifier:** jcooper	**Task Number:** 10084
Pattern: Sequential Workflow		

Update Fields...

Customer Id	BD-SMITH
Id	ORD12345
Order Date	2005-01-17
Order Price	1250.00
Order Status	Pending
Order Comments	All items needed immediately
Ship To	James Smith 123 A Street San Francisco CA 99999 USA
Bill To	James Smith 123 A Street San Francisco CA 99999 USA

User Contact

Phone Number	12223334444
Phone Number	jsmith@companyabc.com

Supplier Info

Supplier Price	1250.00
Supplier Name	BookDepot

Items

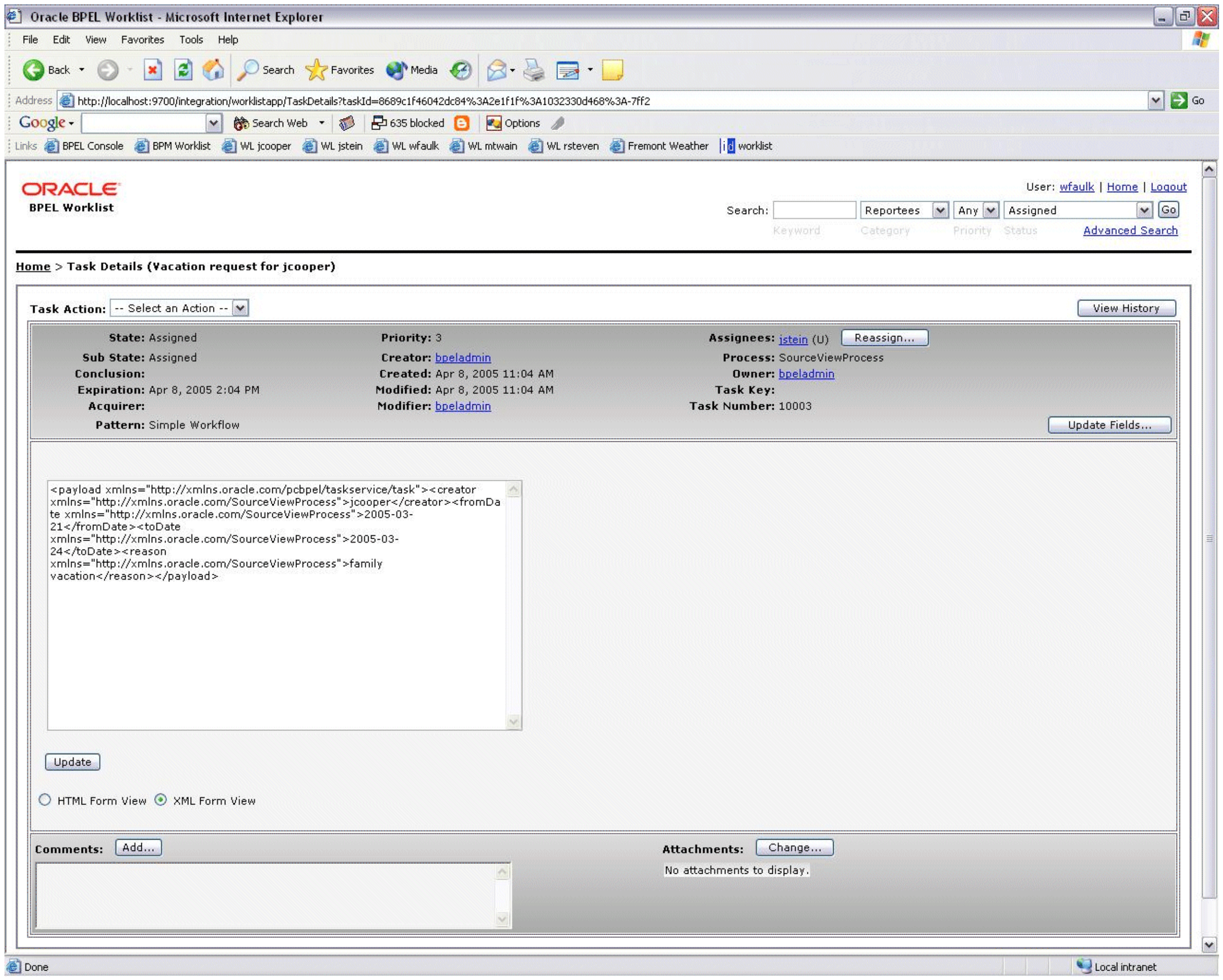
Product Name	Item Type	Part Number	Price	Quantity
Da Vinci Code	Fiction	ISBN123	10.00	50
Harry Potter and the Prisoner of Azkaban	Fiction	ISBN789	10.00	25
Harry Potter and the Order of the Phoenix	Fiction	ISBN456	20.00	25

Comments: Add...   Attachments: Change...

By default the payload display form is automatically generated. As shown in Figure 17–10, it consists of a form containing inputs for each of the elements of the payload and the values corresponding to the original values of the elements when the task was initiated. Using the **Update Fields...** button changes the values.

Figure 17–11 shows a payload in the XML source view. Users can edit the payload in this window, or, for complex payloads, they can copy the XML data into an external XML editor and paste the edited XML back into the XML payload window.

Figure 17–11 Task Payload—XML Source View



Oracle BPEL Worklist - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:9700/integration/worklistapp/TaskDetails?taskId=8689c1f46042dc84%3A2e1f1f%3A1032330d468%3A-7ff2 Go
Google Search Web 635 blocked Options
Links BPEL Console BPM Worklist WL jcooper WL jstein WL wfaulk WL mtwain WL rsteven Fremont Weather worklist
ORACLE BPEL Worklist
User: wfaulk | Home | Logout
Search: Reportees Any Assigned Go
Keyword Category Priority Status Advanced Search
View History
Reassign...
Update Fields...
Update
Add...
Change...
Done
Local intranet

Home > Task Details (Vacation request for jcooper)

Task Action: -- Select an Action --

State: Assigned	**Priority:** 3	**Assignees:** jstein (U)
Sub State: Assigned	**Creator:** bpeladmin	**Process:** SourceViewProcess
Conclusion:	**Created:** Apr 8, 2005 11:04 AM	**Owner:** bpeladmin
Expiration: Apr 8, 2005 2:04 PM	**Modified:** Apr 8, 2005 11:04 AM	**Task Key:**
Acquirer:	**Modifier:** bpeladmin	**Task Number:** 10003
Pattern: Simple Workflow		

```
<payload xmlns="http://xmlns.oracle.com/pcbpel/taskservice/task"><creator
xmlns="http://xmlns.oracle.com/SourceViewProcess">jcooper</creator><fromDa
te xmlns="http://xmlns.oracle.com/SourceViewProcess">2005-03-
21</fromDate><toDate
xmlns="http://xmlns.oracle.com/SourceViewProcess">2005-03-
24</toDate><reason
xmlns="http://xmlns.oracle.com/SourceViewProcess">family
vacation</reason></payload>
```

HTML Form View [✓ XML Form View]

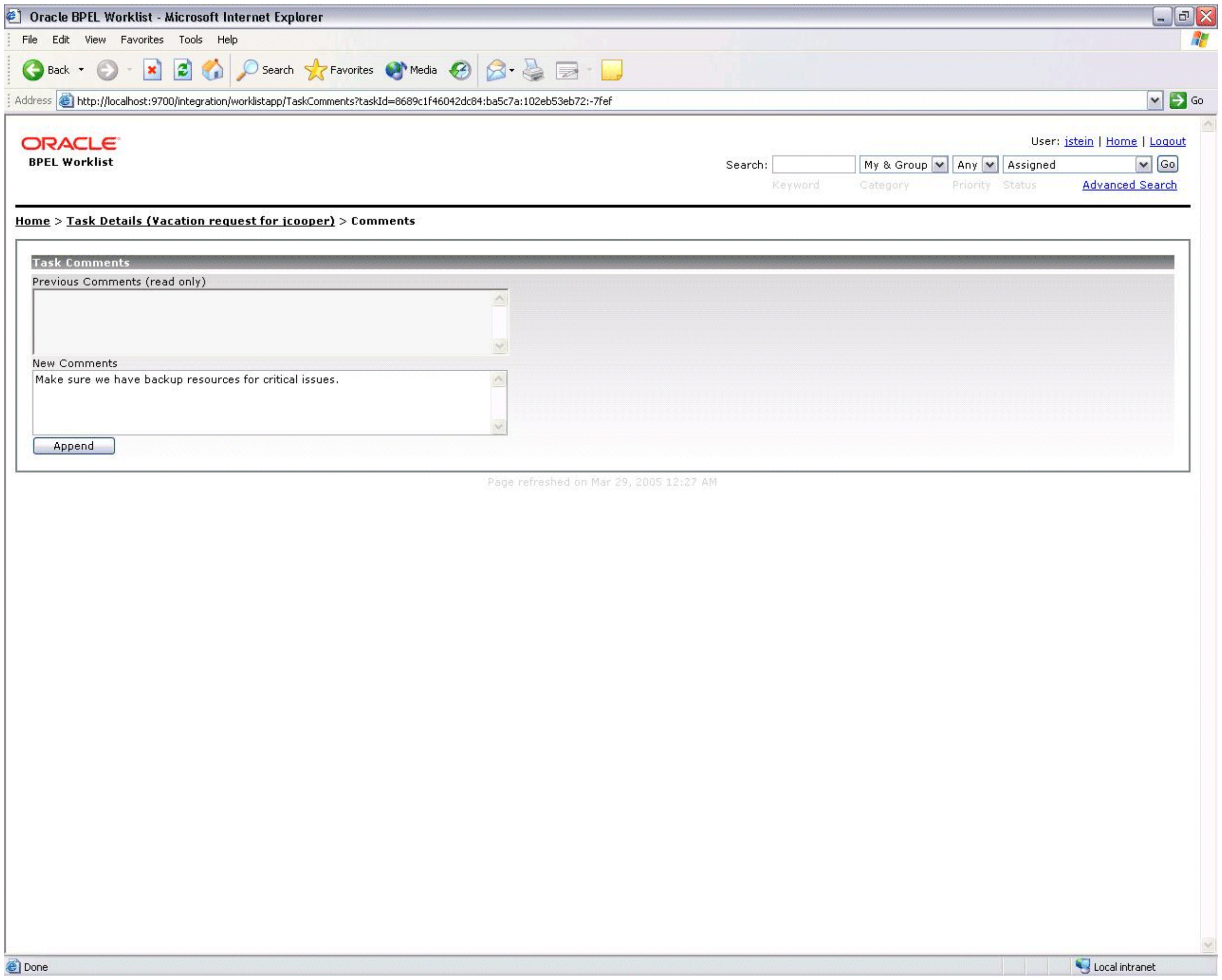
Comments:

Attachments: No attachments to display.

Comments

Figure 17–12 shows where users add comments. To add a comment, users must have permission to update the task.

Figure 17–12 Adding a Comment


Oracle BPEL Worklist - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address
Go
ORACLE
BPEL Worklist
User: jstein | Home | Logout
Search:
My & Group
Any
Assigned
Go
Keyword Category Priority Status
Advanced Search
Append
Done
Local intranet

http://localhost:9700/integration/worklistapp/TaskComments?taskId=8689c1f46042dc84:ba5c7a:102eb53eb72:-7fef

Home > Task Details (Vacation request for jcooper) > Comments

Task Comments

Previous Comments (read only)

New Comments

Make sure we have backup resources for critical issues.

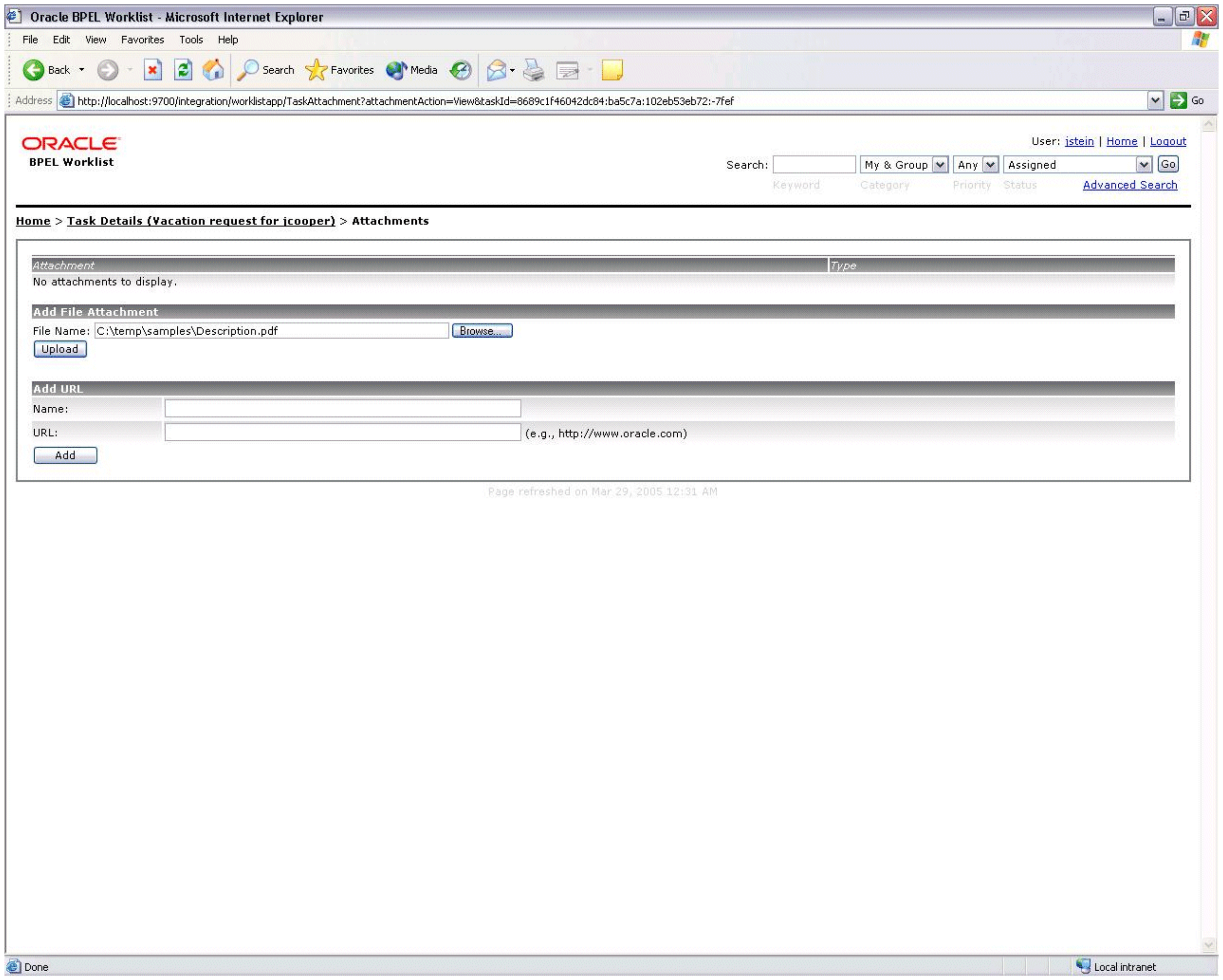
Page refreshed on Mar 29, 2005 12:27 AM

A newly added comment and the comment writer's username are appended to the existing comments, as shown in Figure 17–3 on page 17-9. A trail of comments is maintained throughout the life cycle of the task.

Attachments

Figure 17–13 shows where users add attachments. Users can add a file by using an absolute path name, as shown in the figure, or by using the **Browse** button. Users can associate a URL with a task by providing a name and a well-formed address (for example, `http://www.oracle.com`).

Figure 17–13 Adding Attachments


Oracle BPEL Worklist - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:9700/integration/worklistapp/TaskAttachment?attachmentAction=View&taskId=8689c1f46042dc84:ba5c7a:102eb53eb72:-7fef
ORACLE BPEL Worklist
User: jstein | Home | Logout
Search: My & Group Any Assigned Go
Keyword Category Priority Status Advanced Search

Home > Task Details (Vacation request for jcooper) > Attachments

Attachment	Type
No attachments to display.	

Add File Attachment

File Name: C:\temp\samples\Description.pdf [Browse...]

[Upload]

Add URL

Name:

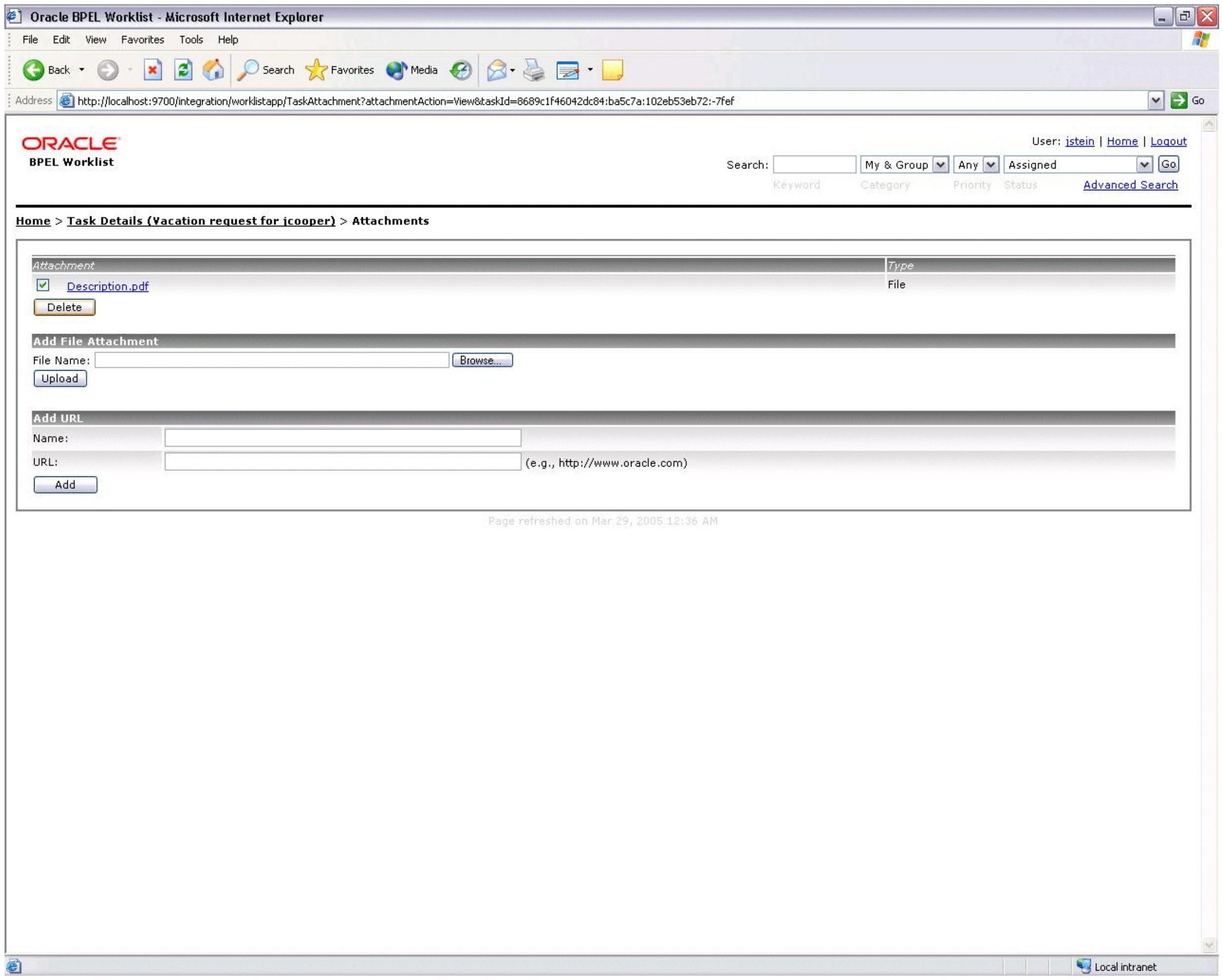
URL: (e.g., http://www.oracle.com)

[Add]

Page refreshed on Mar 29, 2005 12:31 AM

Users can select one or more of the attachments and delete them, as shown in Figure 17–14.

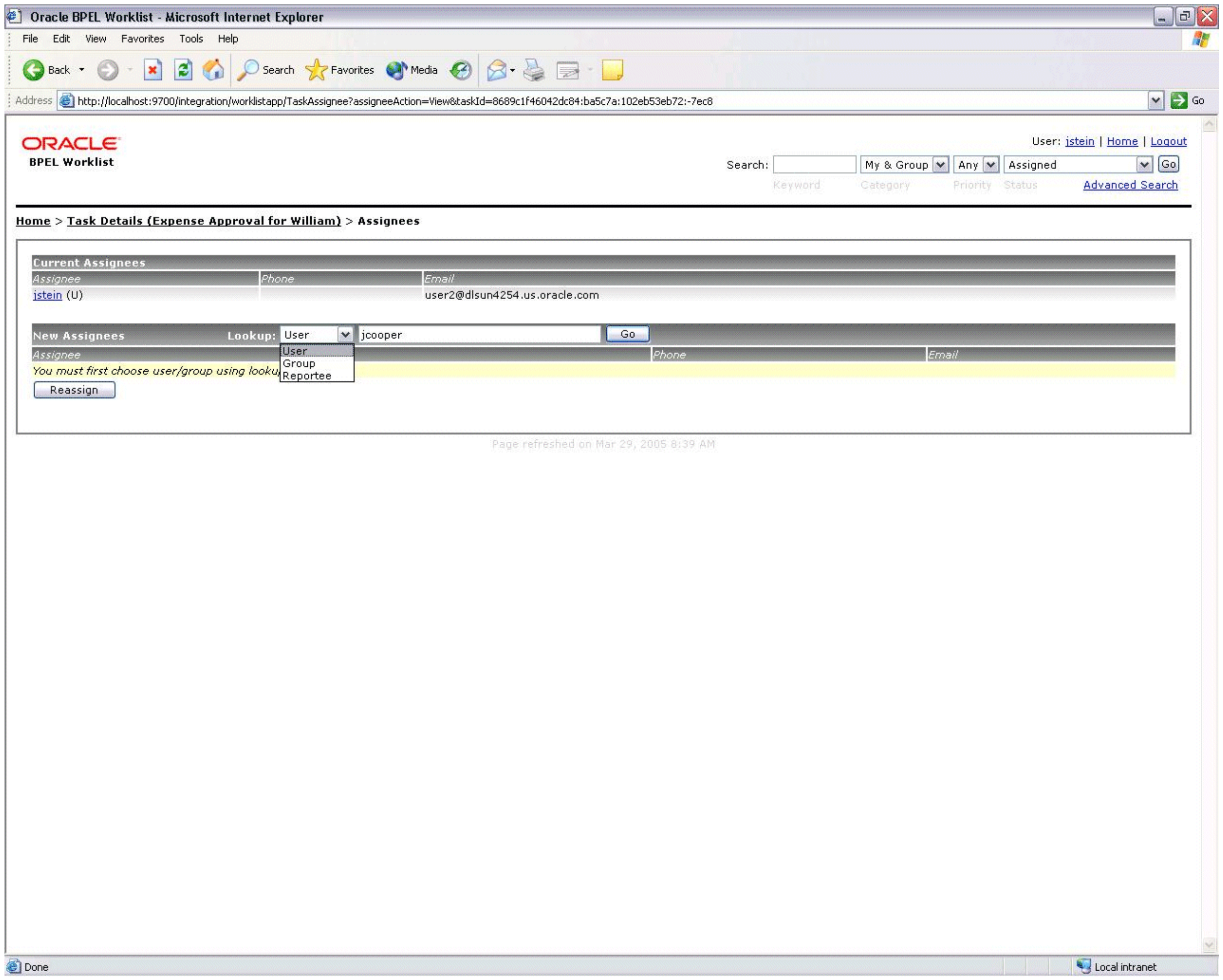
Figure 17–14 Deleting an Attachment


Oracle BPEL Worklist - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:9700/integration/worklistapp/TaskAttachment?attachmentAction=View&taskId=8689c1f46042dc84:ba5c7a:102eb53eb72:-7fef
Go
ORACLE
BPEL Worklist
User: jstein | Home | Logout
Search: My & Group Any Assigned Go
Keyword Category Priority Status Advanced Search
Home > Task Details (Vacation request for jcooper) > Attachments
Attachment Type
Description.pdf File
Delete
Add File Attachment
File Name: Browse...
Upload
Add URL
Name:
URL: (e.g., http://www.oracle.com)
Add
Page refreshed on Mar 29, 2005 12:36 AM
Local intranet

Reassignment

Figure 17–15 shows where users reassign a task.

Figure 17–15 Reassigning a Task


Oracle BPEL Worklist - Microsoft Internet Explorer

http://localhost:9700/integration/worklistapp/TaskAssignee?assigneeAction=View&taskId=8689c1f46042dc84:ba5c7a:102eb53eb72:-7ec8

ORACLE BPEL Worklist

User: jstein | Home | Logout

Search: Keyword | My & Group (Category) | Any (Priority) | Assigned (Status) | Go | Advanced Search

Home > Task Details (Expense Approval for William) > Assignees

Current Assignees

Assignee	Phone	Email
jstein (U)		user2@dlsun4254.us.oracle.com

New Assignees Lookup: User | jcooper | Go

Assignee	Phone	Email
You must first choose user/group using looku		

Reassign

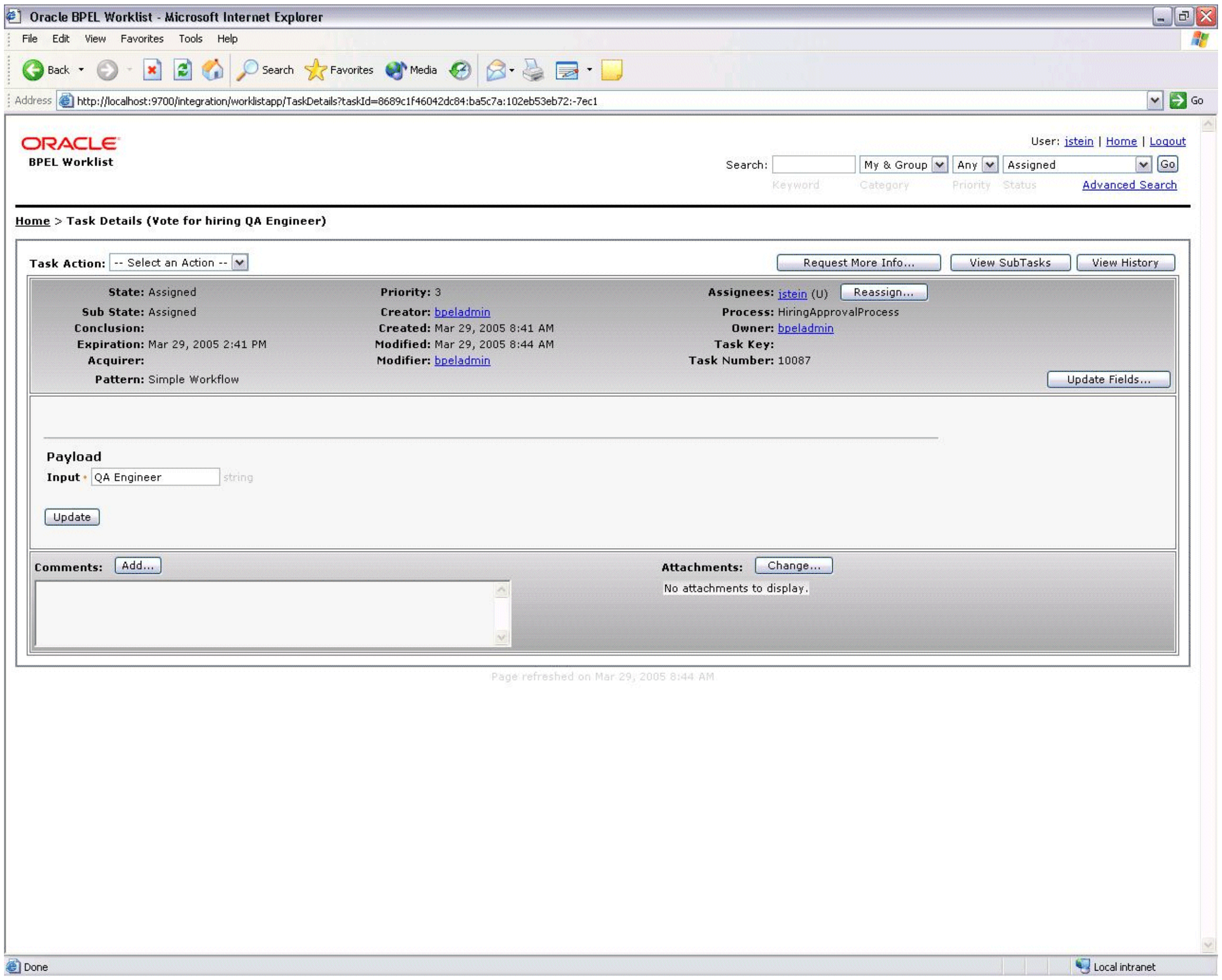
Page refreshed on Mar 29, 2005 8:39 AM

The page lists the current assignees and new assignees. Users can use **Lookup** to find additional names from the JAZN user community. Names that match the search string (wildcard search is supported) are added to the list of new assignees. Deselecting an assignee name drops the user from the list. After a task has been reassigned, it is available in the worklists of the new assignees.

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, as shown in Figure 17–16. Clicking this button displays the SubTasks page, which 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.

Figure 17–16 The View SubTasks Button



http://localhost:9700/integration/worklistapp/TaskDetails?taskId=8689c1f46042dc84:ba5c7a:102eb53eb72:-7ec1

Home > Task Details (Vote for hiring QA Engineer)

Task Action: -- Select an Action --

State: Assigned	**Priority:** 3	**Assignees:** jstein (U)
Sub State: Assigned	**Creator:** bpeladmin	**Process:** HiringApprovalProcess
Conclusion:	**Created:** Mar 29, 2005 8:41 AM	**Owner:** bpeladmin
Expiration: Mar 29, 2005 2:41 PM	**Modified:** Mar 29, 2005 8:44 AM	**Task Key:**
Acquirer:	**Modifier:** bpeladmin	**Task Number:** 10087
Pattern: Simple Workflow		

Payload

Input * QA Engineer string

Comments:

Attachments:

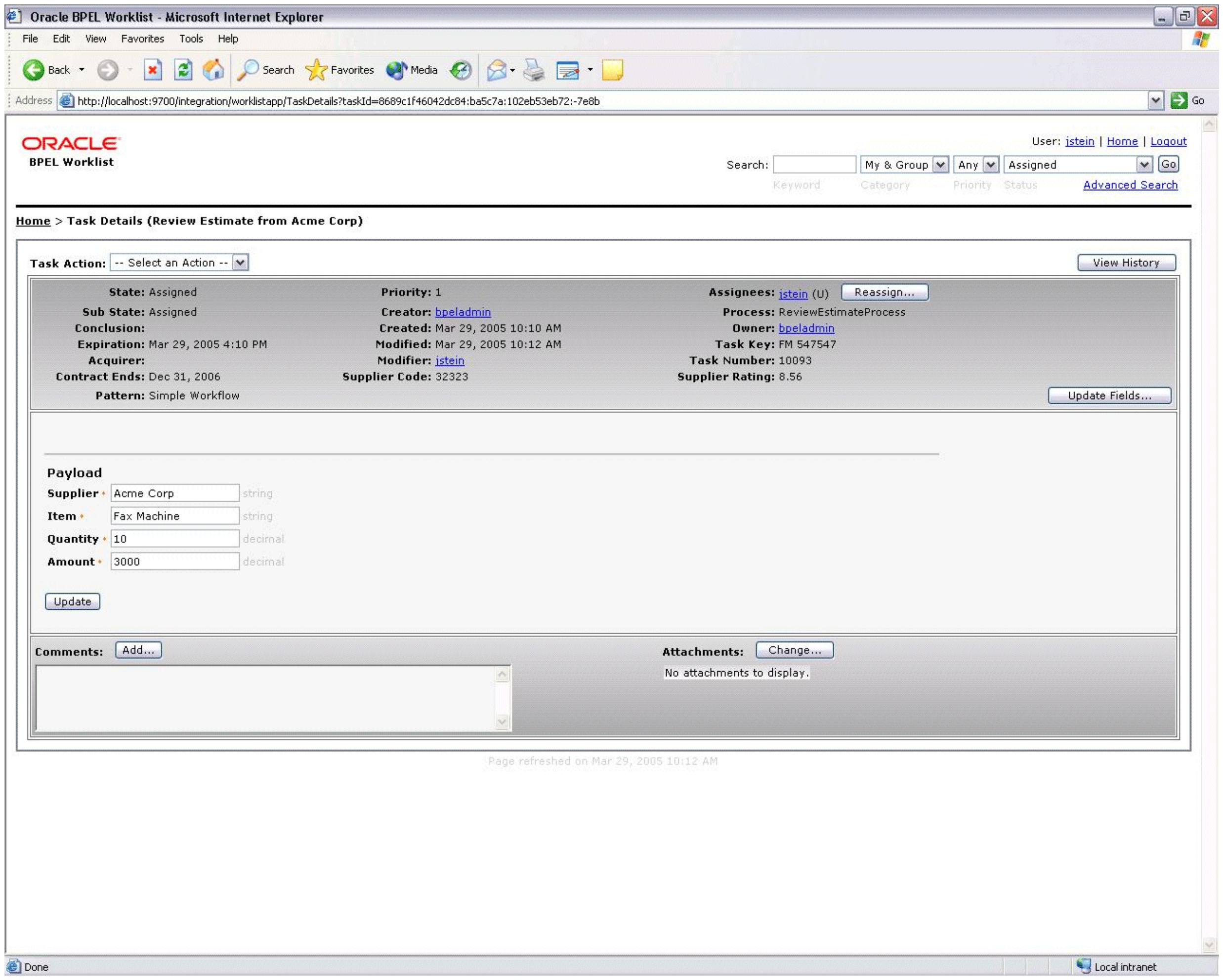
No attachments to display.

Page refreshed on Mar 29, 2005 8:44 AM

Flex Fields and Task Fields Updates

A task can be associated with a set of flex fields of different types. Each flex field can have a custom name. The flex fields are displayed in the Task Details page, as shown in Figure 17–17.

Figure 17–17 Viewing Flex Fields


Oracle BPEL Worklist - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address
Go
ORACLE
BPEL Worklist
User: jstein | Home | Logout
Search:
My & Group
Any
Assigned
Go
Keyword Category Priority Status
Advanced Search
Task Action: -- Select an Action --
View History
Reassign...
Update Fields...
Add...
Change...
Update
Done
Local intranet

http://localhost:9700/integration/worklistapp/TaskDetails?taskId=8689c1f46042dc84:ba5c7a:102eb53eb72:-7e8b

Home > Task Details (Review Estimate from Acme Corp)

State: Assigned
Sub State: Assigned
Conclusion:
Expiration: Mar 29, 2005 4:10 PM
Acquirer:
Contract Ends: Dec 31, 2006
Pattern: Simple Workflow

Priority: 1
Creator: bpeladmin
Created: Mar 29, 2005 10:10 AM
Modified: Mar 29, 2005 10:12 AM
Modifier: jstein
Supplier Code: 32323

Assignees: jstein (U)
Process: ReviewEstimateProcess
Owner: bpeladmin
Task Key: FM 547547
Task Number: 10093
Supplier Rating: 8.56

Payload

Supplier *	Acme Corp	string
Item *	Fax Machine	string
Quantity *	10	decimal
Amount *	3000	decimal

Comments:

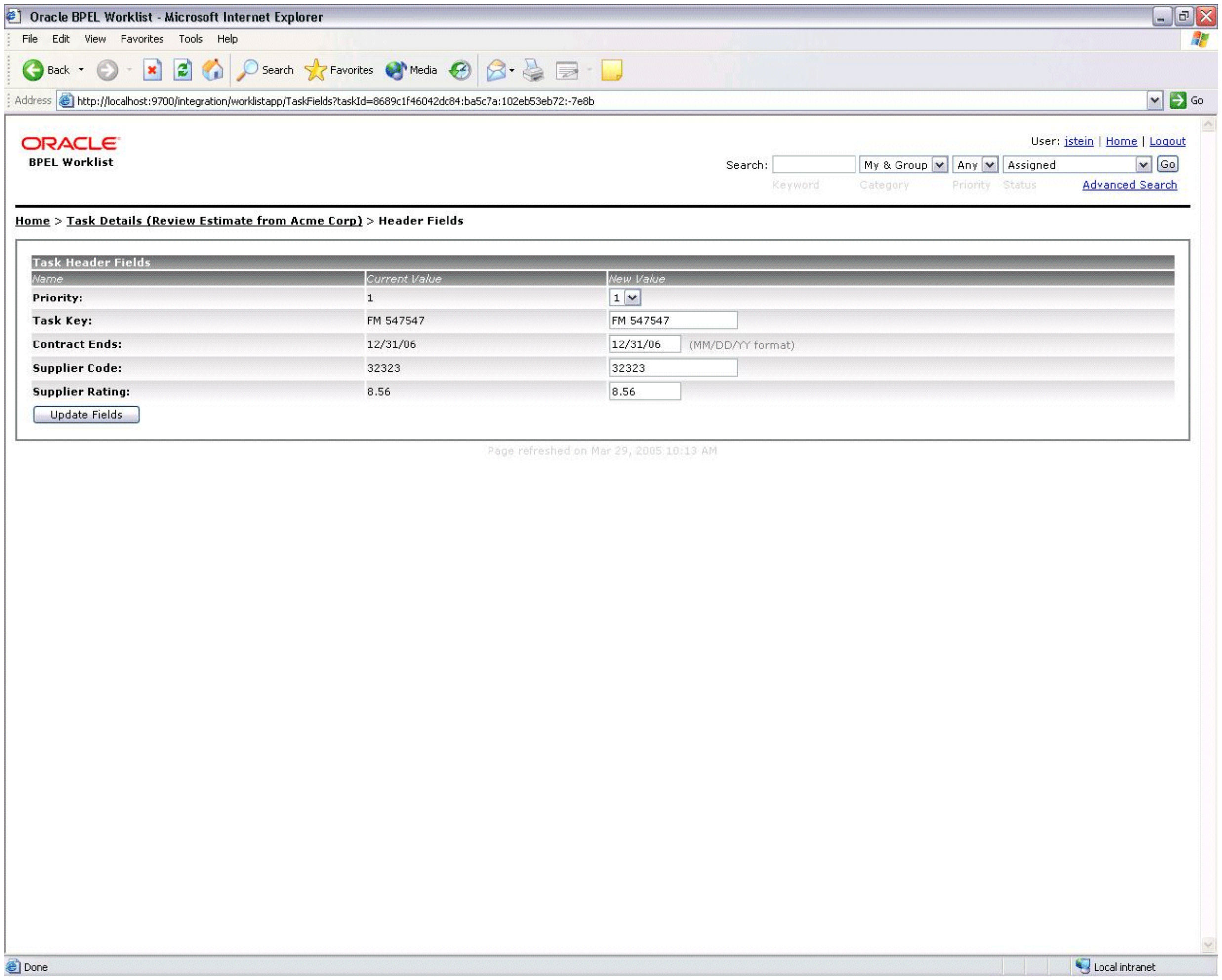
Attachments: No attachments to display.

Page refreshed on Mar 29, 2005 10:12 AM

Clicking **Update Fields...** displays the Header Fields page. This page allows the user to update the flex fields and other task header fields, such as priority and identification key, as shown in Figure 17–18.

See Chapter 16, "Oracle BPEL Process Manager Workflow Services" for information on how flex fields are defined in the advanced option of the workflow wizard.

Figure 17–18 Updating Flex Fields


Oracle BPEL Worklist - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address
Go
ORACLE
BPEL Worklist
User: jstein | Home | Logout
Search:
My & Group
Any
Assigned
Go
Keyword Category Priority Status
Advanced Search
Update Fields
Done
Local intranet

http://localhost:9700/integration/worklistapp/TaskFields?taskId=8689c1f46042dc84:ba5c7a:102eb53eb72:-7e8b

Home > Task Details (Review Estimate from Acme Corp) > Header Fields

Task Header Fields

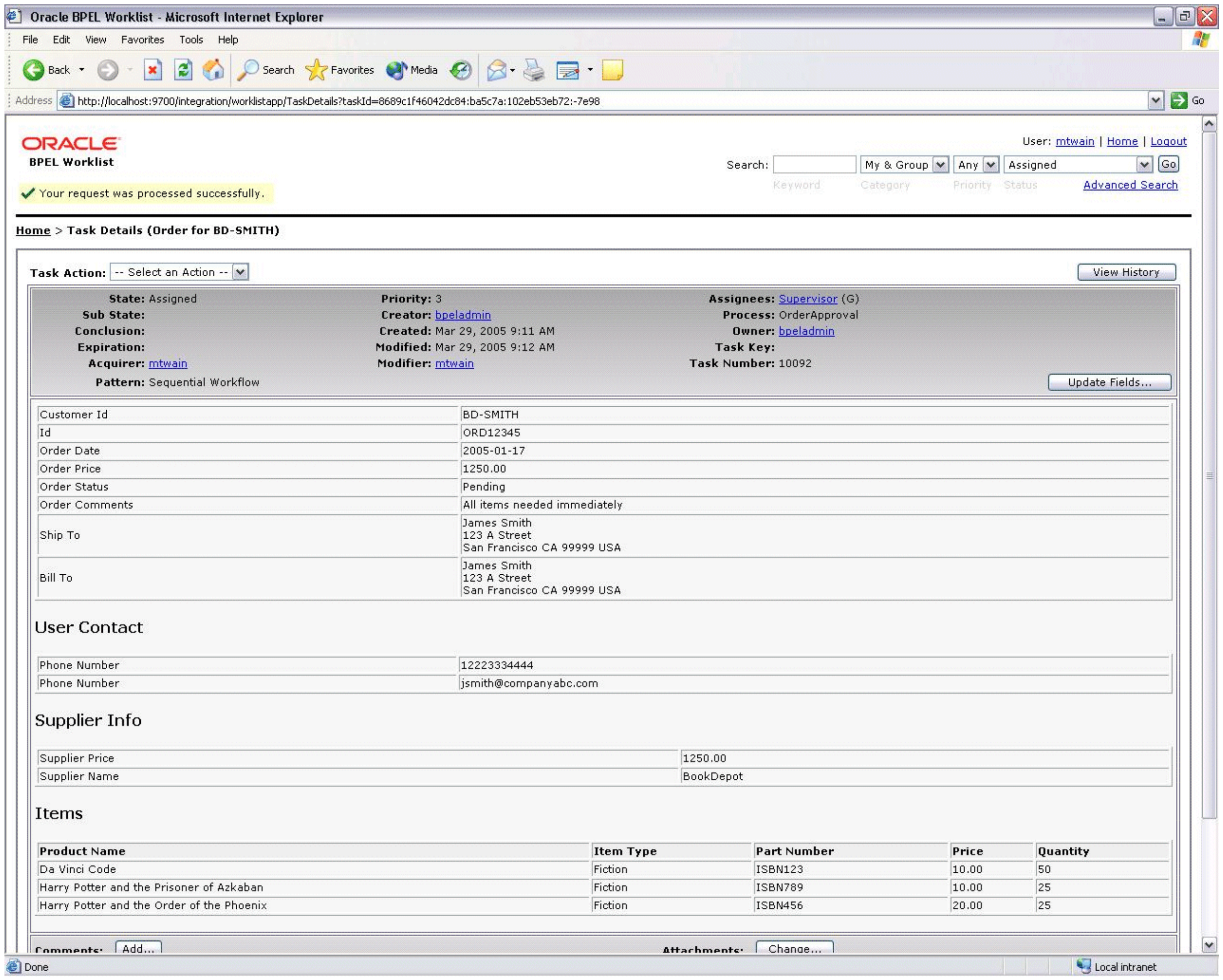
Name	Current Value	New Value
Priority:	1	1
Task Key:	FM 547547	FM 547547
Contract Ends:	12/31/06	12/31/06 (MM/DD/YY format)
Supplier Code:	32323	32323
Supplier Rating:	8.56	8.56

Page refreshed on Mar 29, 2005 10:13 AM

Request Status

For every update request (custom or system action) the user submits, the status of the request is displayed in the left portion of the header. If a request is successful, then the user sees a confirming message, as shown in Figure 17–19.

Figure 17–19 A Successful Update Request


Oracle BPEL Worklist - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:9700/integration/worklistapp/TaskDetails?taskId=8689c1f46042dc84:ba5c7a:102eb53eb72:-7e98
ORACLE BPEL Worklist
User: mtwain | Home | Logout
Search: My & Group Any Assigned Go
Keyword Category Priority Status Advanced Search
Task Action: -- Select an Action --
View History
Update Fields...
Comments: Add...
Attachments: Change...
Done
Local intranet

Your request was processed successfully.

Home > Task Details (Order for BD-SMITH)

State: Assigned
Sub State:
Conclusion:
Expiration:
Acquirer: mtwain
Pattern: Sequential Workflow
Priority: 3
Creator: bpeladmin
Created: Mar 29, 2005 9:11 AM
Modified: Mar 29, 2005 9:12 AM
Modifier: mtwain
Assignees: Supervisor (G)
Process: OrderApproval
Owner: bpeladmin
Task Key:
Task Number: 10092

Customer Id	BD-SMITH
Id	ORD12345
Order Date	2005-01-17
Order Price	1250.00
Order Status	Pending
Order Comments	All items needed immediately
Ship To	James Smith 123 A Street San Francisco CA 99999 USA
Bill To	James Smith 123 A Street San Francisco CA 99999 USA

User Contact

Phone Number	12223334444
Phone Number	jsmith@companyabc.com

Supplier Info

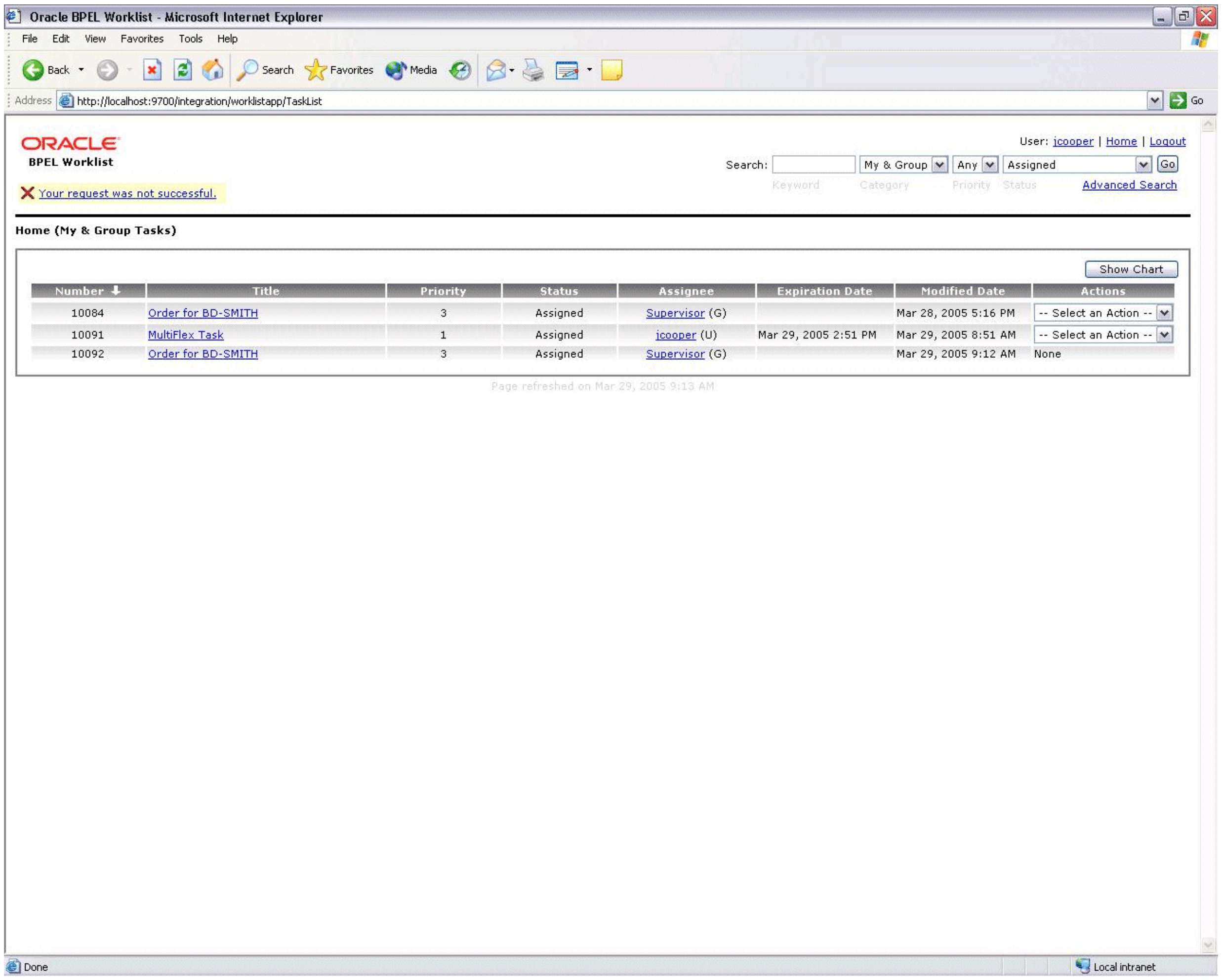
Supplier Price	1250.00
Supplier Name	BookDepot

Items

Product Name	Item Type	Part Number	Price	Quantity
Da Vinci Code	Fiction	ISBN123	10.00	50
Harry Potter and the Prisoner of Azkaban	Fiction	ISBN789	10.00	25
Harry Potter and the Order of the Phoenix	Fiction	ISBN456	20.00	25

If a request is not successful, then the user sees a failure message, as shown in Figure 17–20. Clicking the error message displays the Error Information page. See "Error Information" on page 17-28 for more information.

Figure 17–20 An Unsuccessful Update Request


Oracle BPEL Worklist - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:9700/integration/worklistapp/TaskList
ORACLE
BPEL Worklist
User: jcooper | Home | Logout
Search: My & Group Any Assigned Go
Keyword Category Priority Status Advanced Search
Show Chart
Done
Local intranet

Your request was not successful.

Home (My & Group Tasks)

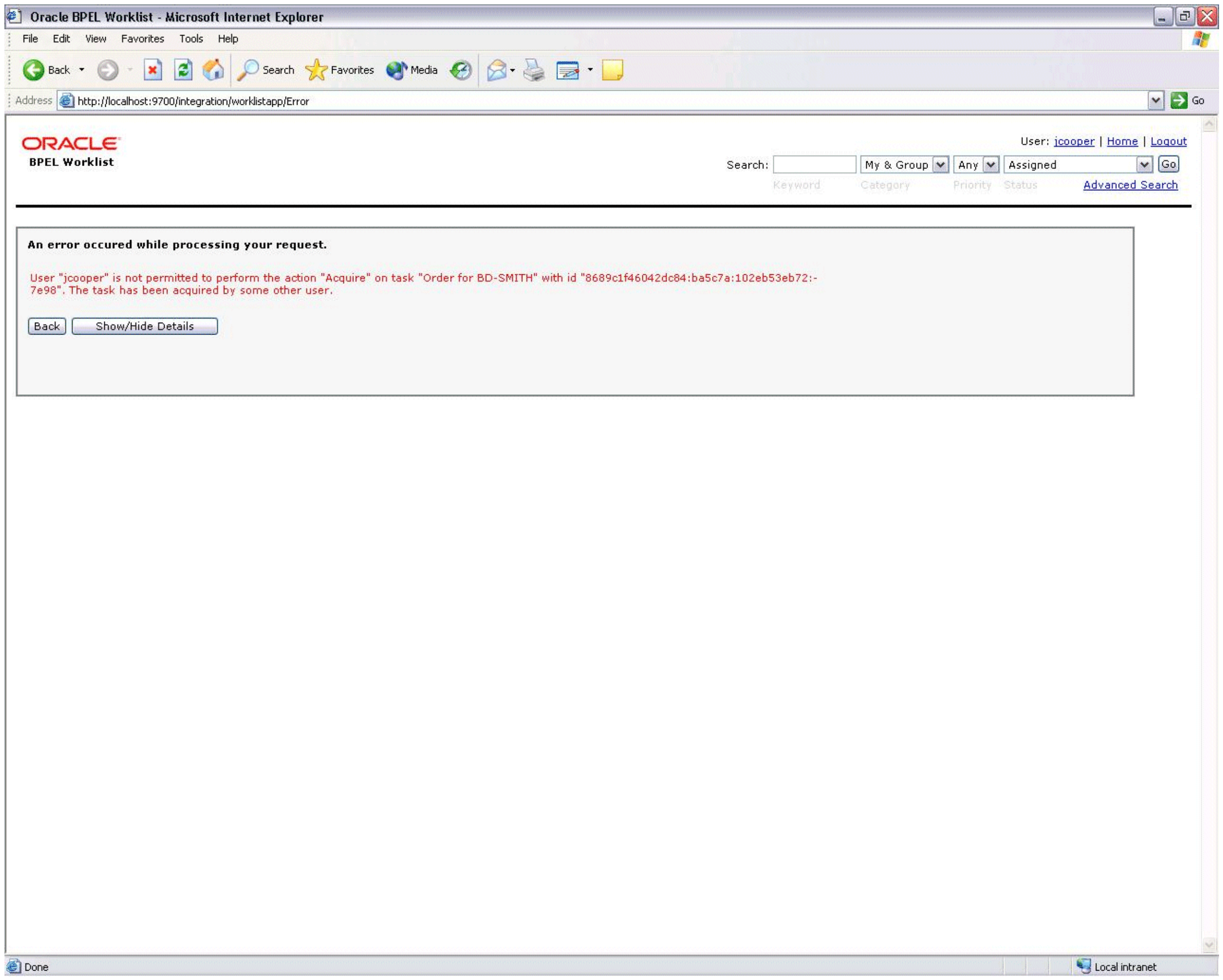
Number	Title	Priority	Status	Assignee	Expiration Date	Modified Date	Actions
10084	Order for BD-SMITH	3	Assigned	Supervisor (G)		Mar 28, 2005 5:16 PM	-- Select an Action --
10091	MultiFlex Task	1	Assigned	jcooper (U)	Mar 29, 2005 2:51 PM	Mar 29, 2005 8:51 AM	-- Select an Action --
10092	Order for BD-SMITH	3	Assigned	Supervisor (G)		Mar 29, 2005 9:12 AM	None

Page refreshed on Mar 29, 2005 9:13 AM

Error Information

If an error is encountered while processing a request, an error message is displayed, as shown in Figure 17–21.

Figure 17–21 Error Message Display


Oracle BPEL Worklist - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:9700/integration/worklistapp/Error
Go
ORACLE
BPEL Worklist
User: jcooper | Home | Logout
Search: My & Group Any Assigned Go
Keyword Category Priority Status Advanced Search
Back
Show/Hide Details
Done
Local intranet

An error occured while processing your request.

User "jcooper" is not permitted to perform the action "Acquire" on task "Order for BD-SMITH" with id "8689c1f46042dc84:ba5c7a:102eb53eb72:-7e98". The task has been acquired by some other user.

The main message of the error is displayed by default. Click **Show/Hide Details** for details (mostly useful for application developers or administrators). Clicking **Back** displays the previous page where the request was made.

The user error shown in Figure 17–21 occurs when a user has attempted an action that is not permitted. This is possible in the following scenarios:

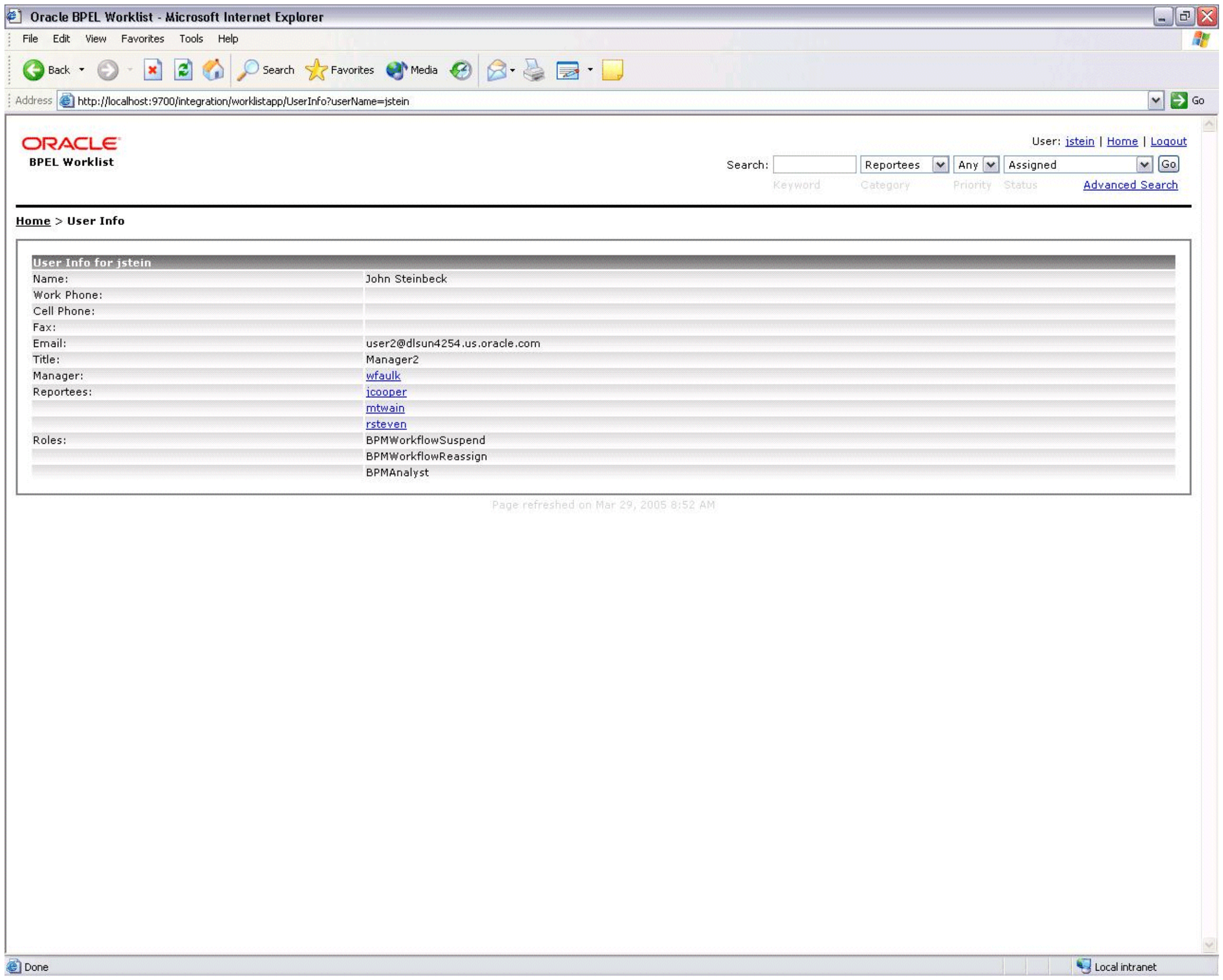
- The task expired between the time the user loaded the page and actually performed the action.
- The task was acquired and updated concurrently by another user (such as a manager, owner, or administrator) between the time the user loaded the page and actually performed the action.

The Error Message page also displays system errors, which should be reported to the administrator.

User and Group Information

If a user clicks the link in the header that identifies the logged-in user, the User Info page appears. This page displays 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 17–22.

Figure 17–22 User Information

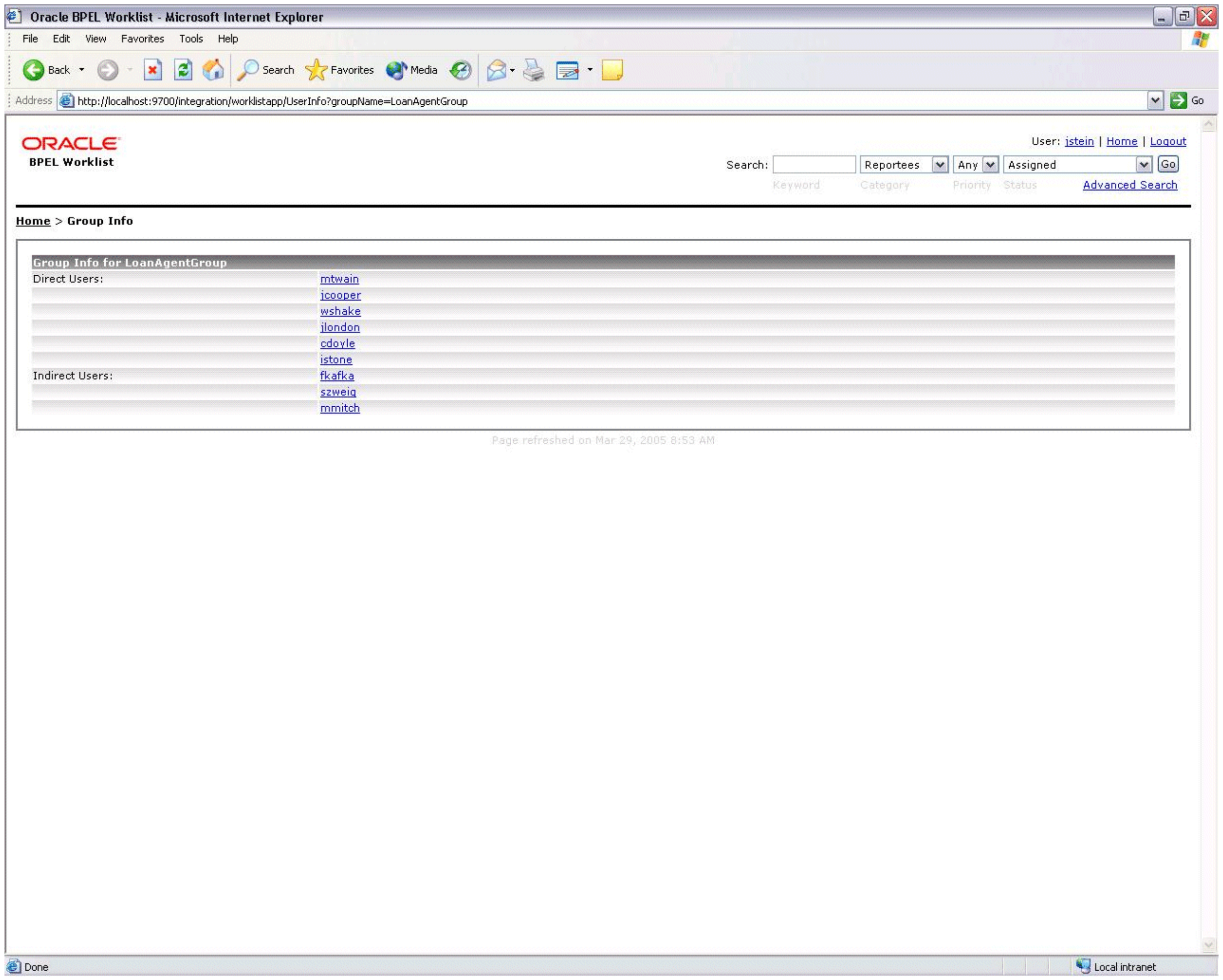

Oracle BPEL Worklist - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:9700/integration/worklistapp/UserInfo?userName=jstein
ORACLE BPEL Worklist
User: jstein | Home | Logout
Search: Reportees Any Assigned Go
Keyword Category Priority Status Advanced Search
Home > User Info
Done
Local intranet

User Info for jstein	
Name:	John Steinbeck
Work Phone:	
Cell Phone:	
Fax:	
Email:	user2@dlsun4254.us.oracle.com
Title:	Manager2
Manager:	wfaulk
Reportees:	jcooper
	mtwain
	rsteven
Roles:	BPMWorkflowSuspend
	BPMWorkflowReassign
	BPMAnalyst

Page refreshed on Mar 29, 2005 8:52 AM

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, as shown in Figure 17–23. The Group Info page displays the list of direct and indirect users (users contained in child groups of the current group).

Figure 17–23 Group Information


Oracle BPEL Worklist - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Search
Favorites
Media
Address
http://localhost:9700/integration/worklistapp/UserInfo?groupName=LoanAgentGroup
Go
ORACLE
BPEL Worklist
User: jstein | Home | Logout
Search:
Reportees
Any
Assigned
Go
Keyword
Category
Priority
Status
Advanced Search
Home > Group Info
Done
Local intranet

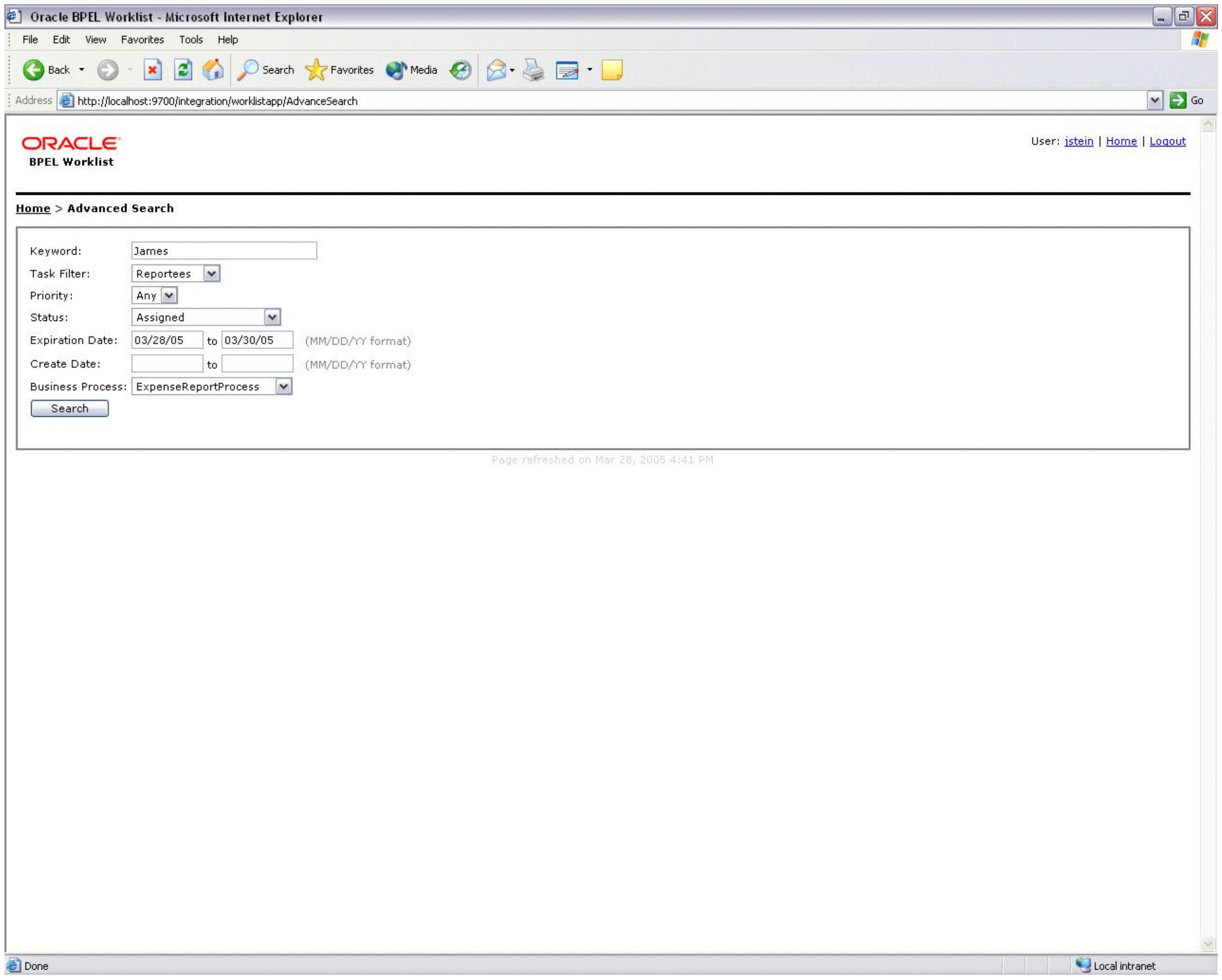
Group Info for LoanAgentGroup	
Direct Users:	mtwain
	jcooper
	wshake
	jlondon
	cdoyle
	istone
Indirect Users:	fkafka
	szweig
	mmitch

Page refreshed on Mar 29, 2005 8:53 AM

Advanced Search

The Advanced Search page, shown in Figure 17–24, provides additional filters for performing a fine-grained search, based on business process, expiration date, and creation date. The standard search box that appears in the top-right corner is not displayed in the advanced search option.

Figure 17–24 Advanced Search Page


Oracle BPEL Worklist - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Search
Favorites
Media
Address
Go
ORACLE
BPEL Worklist
User: jstein | Home | Logout
Home > Advanced Search
Search
Done
Local intranet

http://localhost:9700/integration/worklistapp/AdvanceSearch

Keyword:	James		
Task Filter:	Reportees		
Priority:	Any		
Status:	Assigned		
Expiration Date:	03/28/05	to 03/30/05	(MM/DD/YY format)
Create Date:		to	(MM/DD/YY format)
Business Process:	ExpenseReportProcess		

Page refreshed on Mar 28, 2005 4:41 PM

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 acquire, suspend, or reassign, the worklist 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 has to be acquired first before any actions can be performed on it.

See "Identity Service" on page 16-75 for information about the identity service and how privileges can be assigned to users.

How Changes to a Workflow Appear in the Worklist Application

When a BPEL process is aborted, associated tasks are marked as **Stale** in the **Status** column of the Worklist Application home page. When a BPEL process instance is deleted, all associated tasks are deleted. When a BPEL process is undeployed, associated tasks are marked as **Stale** in the **Status** column of the Worklist Application home page.

Accessing the Worklist Application in Local Languages

The identity service determines a user's preferred locale (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 locale and itemizing for the user is set to the system default.

For example, if you are using the sample worklist configured with the user community in the JAZN XML file, then you can set the user's preferred language in `users-properties.xml` (in *Oracle_Home*`\integration\orabpel\system\services\config`) as follows:

```
<bpm:timeZone>America/Los_Angeles</bpm:timeZone>
<bpm:preferredLanguage>en_US</bpm:preferredLanguage>
```

If an LDAP-based provider such as OID is used, then language settings 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. For custom actions, flex fields, and task titles, the display names come from the message bundle, if specified in the configuration. If no message bundle is specified, then the values specified in the wizard at design time are used. See Chapter 16, "Oracle BPEL Process Manager Workflow Services" for more information on how to specify message bundles so that custom actions, flex fields, and task titles are displayed in a preferred local language.

The Worklist Application supports the locales shown in Table 17–4.

Table 17–4 Languages and Java Locales Supported by the Worklist Application

Language	Java Locale
English	(en)
English [United States]	(en_US)
German	(de)
Spanish [International]	(es)
Spanish [Spain]	(es_ES)

Table 17–4 (Cont.) Languages and Java Locales Supported by the Worklist Application

Language	Java Locale
French	(fr)
French [Canada]	(fr_CA)
Italian	(it)
Japanese	(ja)
Korean	(ko)
Portuguese	(pt)
Portuguese [Brazil]	(pt_BR)
Chinese [Simplified]	(zh_CN)
Chinese [Traditional]	(zh_TW)

Customizing the Worklist Application

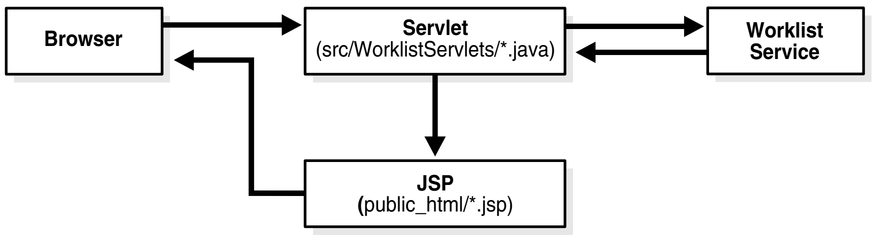
The sample Worklist Application described in this chapter is fully functional. Use it as a starting point to create a customized Worklist Application to suit your specific needs. This section discusses the architecture of the Worklist Application and provides specific details for customizing it.

See Also: The `WorklistServlets` directory in *`Oracle_Home`*`\integration\orabpel\samples\hw\worklistxpress\src`

Worklist Application Architecture

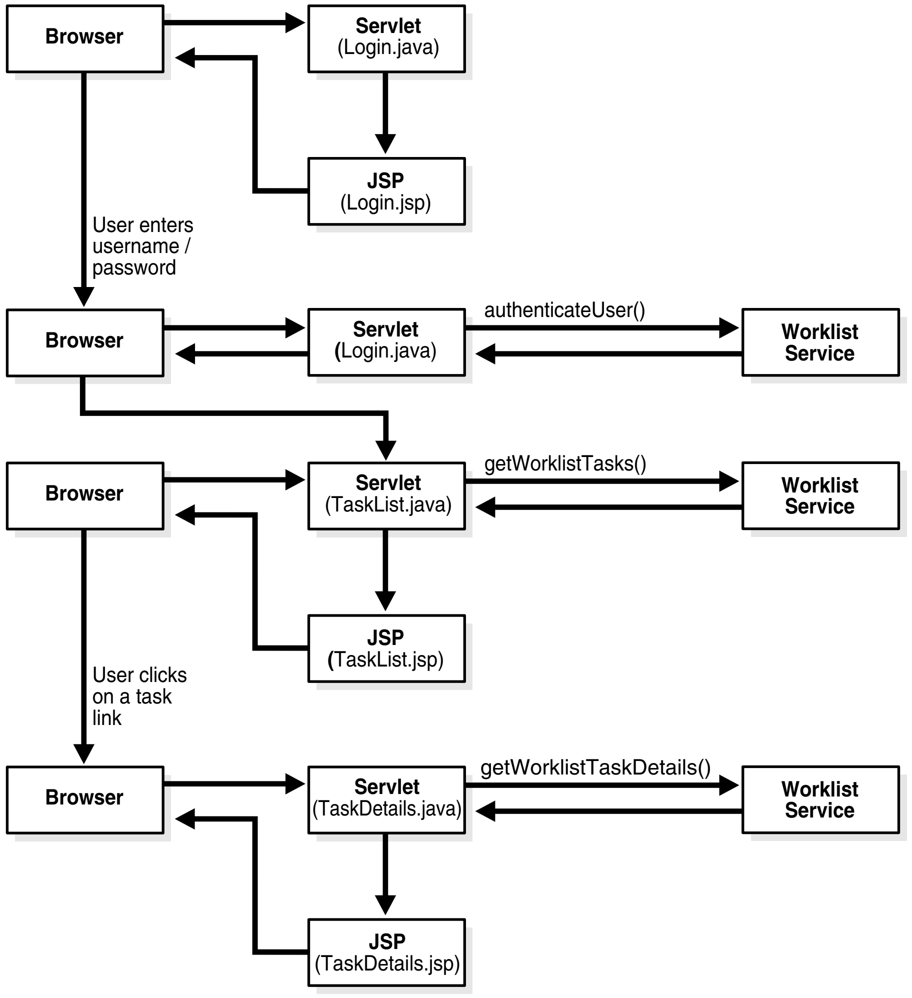
The Worklist Application follows the standard model-view-controller approach, as shown in Figure 17–25. A request coming from the browser is handled by a servlet. The servlet validates the request and calls the appropriate worklist service API to either fetch the required data from the back end or to update a task. After the API call, the servlet stores the data required for rendering the next page in the session. The JSP page picks up the data from the session, renders the data, and removes it from the session. Hence the servlets and the JSP pages have different functions. The servlets are responsible for making the back-end API calls and the JSP pages are responsible for formatting the data.

Figure 17–25 Worklist Application Architecture



A typical page flow sequence is shown in Figure 17–26. 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 performs a page redirect to the login JSP page that is sent to the browser. The user enters the username and password and the login servlet calls the `authenticateUser()` API. If successful, it sends a redirect to the TaskList URL. The browser's request then goes to the TaskList servlet that calls the `getWorklistTasks()` API for getting the tasks that the user should see. Then it performs a page redirect to the TaskList JSP page that is sent to the browser. When a user clicks a task link, the request is handled by the TaskDetails servlet. This calls the `getWorklistTaskDetails()` API and performs a page redirect to the TaskDetails JSP page that is sent to the browser. Page flows for other functionality, such as updating payload, changing priority, adding an attachment, reassigning a task, viewing history, and requesting more information, are similar.

Figure 17–26 A Typical Page Flow Sequence



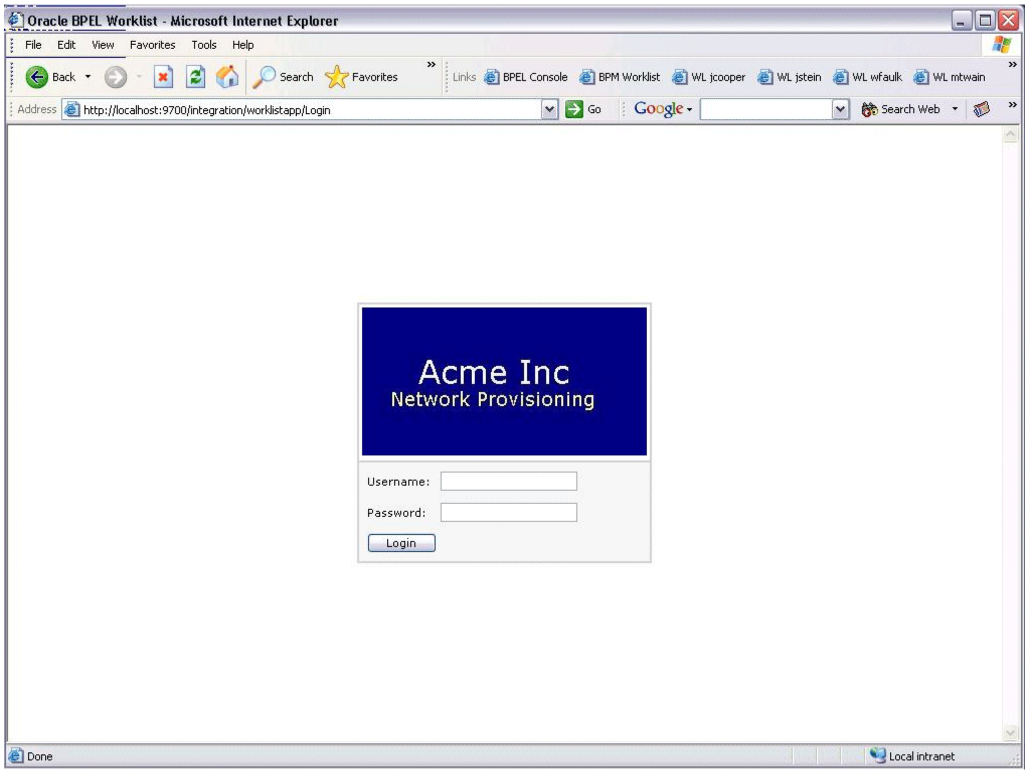
The separation of responsibility facilitates customizing the application. The page flow requirements for many customer requirements are probably similar to the page flow for the sample Worklist Application. Therefore, it may be sufficient to modify the JSP pages (and the Java class `HTMLFormatter.java` used for formatting HTML data).

The following sections discuss how to customize some of the commonly used pages for a different look and feel to suit various application requirements.

Login Page

A common requirement for customizing the login page is to add corporate branding, as shown in Figure 17–27. Replace the image tag `<img src="img/login.jpg">` in the `Login.jsp` page with your image tag, such as `<img src="img/acmeLogin.jpg">`. Everything else remains the same.

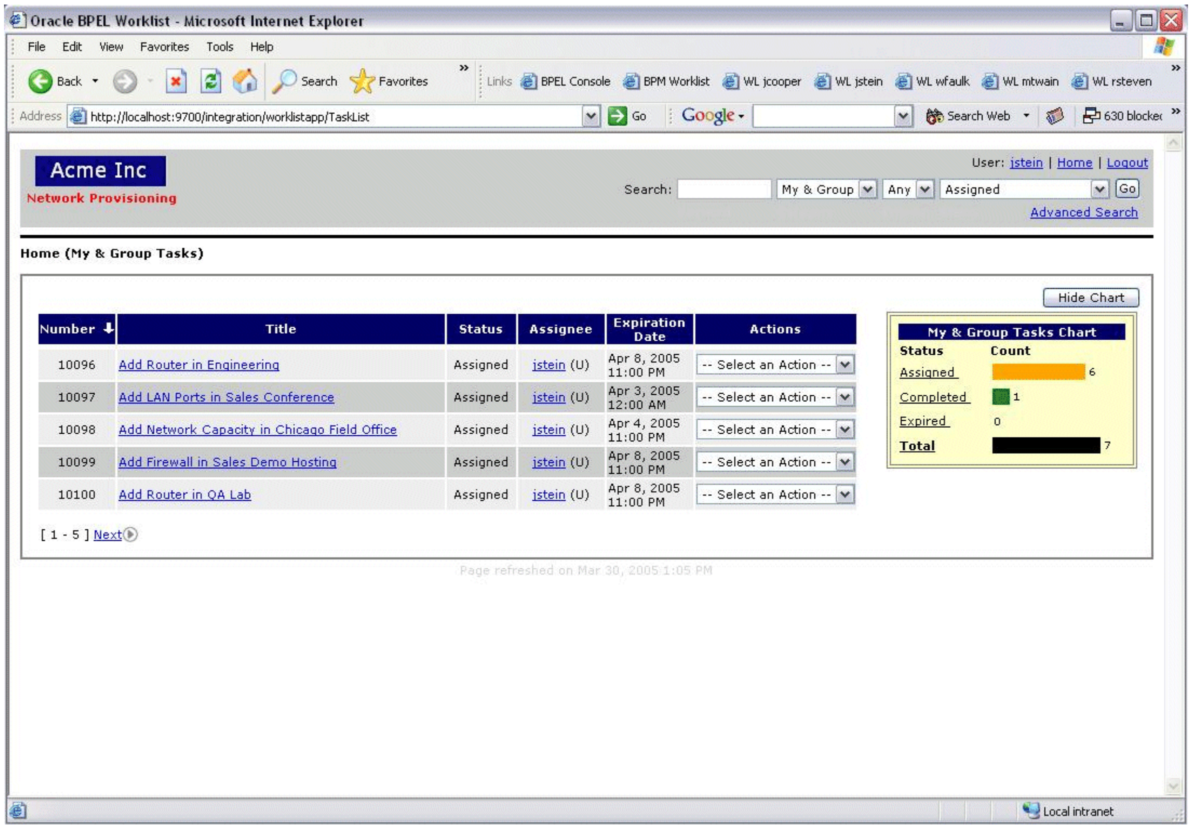
Figure 17–27 Login Page


Oracle BPEL Worklist - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Links BPEL Console BPM Worklist WL jcooper WL jstein WL wfaulk WL mtwain
Address http://localhost:9700/integration/worklistapp/Login
Go
Google
Search Web
Acme Inc
Network Provisioning
Username:
Password:
Login
Done
Local intranet

Header Info

The header section appears on every page above the bread crumb navigation. This section can be customized by modifying the `Header.jsp` file, as shown in Figure 17–28. 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.

Figure 17–28 Header Information


Oracle BPEL Worklist - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Links BPEL Console BPM Worklist WL jcooper WL jstein WL wfaulk WL mtwain WL rsteven
Address http://localhost:9700/integration/worklistapp/TaskList Go
Search Web 630 blocked
Acme Inc
Network Provisioning
User: jstein | Home | Logout
Search: My & Group Any Assigned Go
Advanced Search
Hide Chart
Local intranet

Home (My & Group Tasks)

Number	Title	Status	Assignee	Expiration Date	Actions
10096	Add Router in Engineering	Assigned	jstein (U)	Apr 8, 2005 11:00 PM	-- Select an Action --
10097	Add LAN Ports in Sales Conference	Assigned	jstein (U)	Apr 3, 2005 12:00 AM	-- Select an Action --
10098	Add Network Capacity in Chicago Field Office	Assigned	jstein (U)	Apr 4, 2005 11:00 PM	-- Select an Action --
10099	Add Firewall in Sales Demo Hosting	Assigned	jstein (U)	Apr 8, 2005 11:00 PM	-- Select an Action --
10100	Add Router in QA Lab	Assigned	jstein (U)	Apr 8, 2005 11:00 PM	-- Select an Action --

[1 - 5] Next

My & Group Tasks Chart

Status	Count
Assigned	6
Completed	1
Expired	0
Total	7

Page refreshed on Mar 30, 2005 1:05 PM

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.

Task Home (Listing) Page

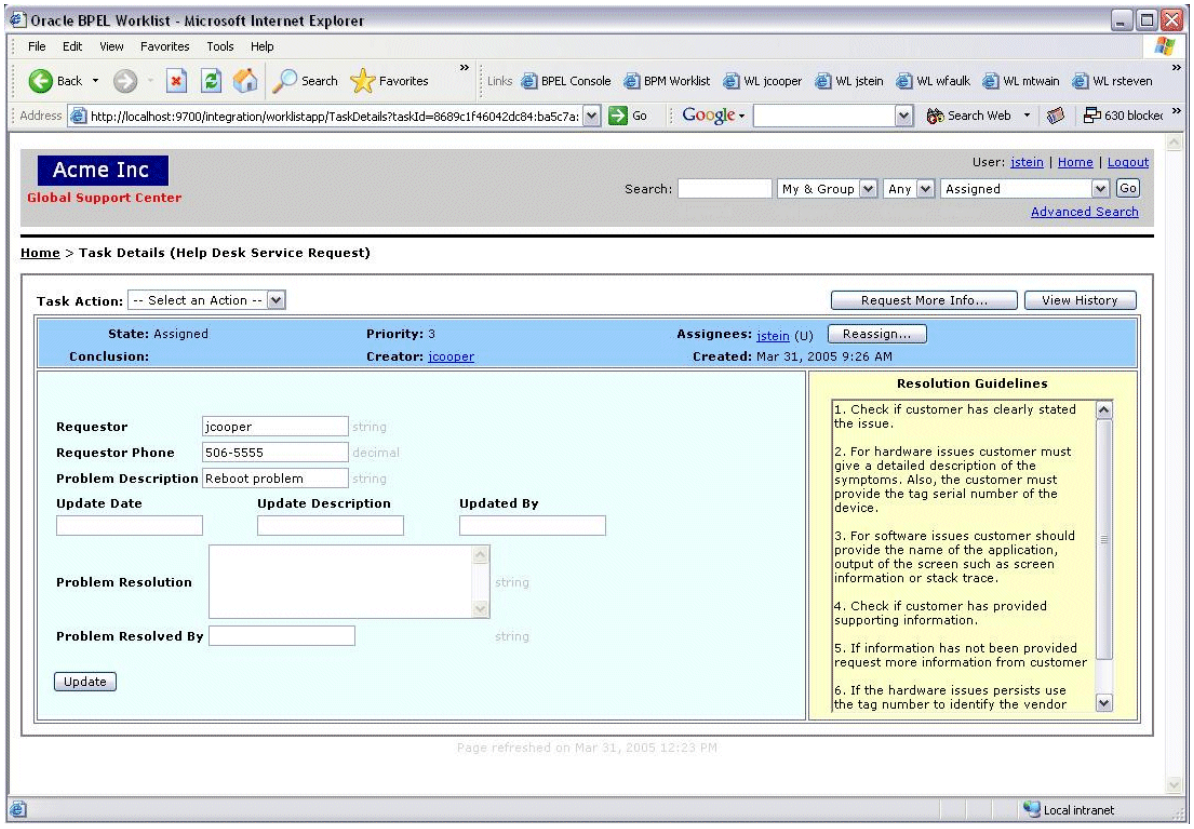
The home (or listing) page lists all the tasks that match the search criteria specified by the user. It also contains a chart that summarizes the task counts for the category (task) filter. As shown in Figure 17–28, both the listing and the chart content can be customized to suit application requirements. You can customize the list by modifying the `displayFilteredTasks()` method in the `src/WorklistServlets/HTMLFormatter.java` class. Some of the formatting code is included in this class for modularity and reuse. You can customize the list to display only those columns that are relevant to the application. The contents of the columns can be customized as well. You can customize the list of task actions by modifying the `displayTaskActionsDropdown()` method in the same class. You can customize the chart, for example, to show only counts for statuses that are relevant to the application by modifying the `ListChart.jsp` file. Customize the overall layout by modifying the `TaskList.jsp` file.

Task Details Page

The Task Details page is typically 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. All of these can be customized by modifying the `TaskDetails.jsp` file.

In the example shown in Figure 17–29, the header section can be customized to display only the task fields that are relevant to the application. The payload section contains the autogenerated JSP file based on the payload XSD. See Chapter 16, "Oracle BPEL Process Manager Workflow Services" for information on how to customize this file. The footer section can be customized, moved to the top section, or eliminated, depending on the application requirements.

Figure 17–29   Task Details Page


Oracle BPEL Worklist - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Links BPEL Console BPM Worklist WL jcooper WL jstein WL wfaulk WL mtwain WL rsteven
Address http://localhost:9700/integration/worklistapp/TaskDetails?taskId=8689c1f46042dc84:ba5c7a: Go
Google Search Web 630 blocked
Acme Inc
Global Support Center
User: jstein | Home | Logout
Search: My & Group Any Assigned Go
Advanced Search
Home > Task Details (Help Desk Service Request)
Task Action: -- Select an Action --
Request More Info...
View History
Reassign...
Update
Local intranet

State: Assigned
Conclusion:
Priority: 3
Creator: jcooper
Assignees: jstein (U)
Created: Mar 31, 2005 9:26 AM

Requestor	jcooper	string
Requestor Phone	506-5555	decimal
Problem Description	Reboot problem	string

Update Date | Update Description | Updated By

Problem Resolution — string

Problem Resolved By — string

Resolution Guidelines

1. Check if customer has clearly stated the issue.

2. For hardware issues customer must give a detailed description of the symptoms. Also, the customer must provide the tag serial number of the device.

3. For software issues customer should provide the name of the application, output of the screen such as screen information or stack trace.

4. Check if customer has provided supporting information.

5. If information has not been provided request more information from customer

6. If the hardware issues persists use the tag number to identify the vendor

Page refreshed on Mar 31, 2005 12:23 PM

Additional Pages

The guidelines provided in "Worklist Application Architecture" on page 17-34 can be used to modify other pages to suit the application requirements.

Configuration Parameters

In addition to customizing pages, you can modify parameters in *`Oracle_home`*`\integration\orabpel\system\services\config\pc.properties` to change the behavior of the application as follows:

- To change the page size on the listing page, change:

  ```
  oracle.tip.worklist.samples.tasklist.maxrows=5
  ```

- To change the renewal duration for tasks, change:

  ```
  oracle.tip.worklist.samples.taskaction.renew.duration=P7D
  ```

Controlling Access to Information and Actions for Different Users

The worklist 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) have 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:

- `ACQUIRE`
- `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.

The Worklist Application can be customized so that the information viewed and the actions performed on a given page can be 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 page, the identity service can be used to check if the user has the granted role and determine which code path to take.

For example, you can create a new role called `BPMProcessingManager` in the file `jazn-data.xml` (in *`Oracle_Home`* `\integration\orabpel\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
{
  BPMIdentityService mIdentityService =
             ServiceFactory.getIdentityServiceInstance();
  // lookup user based on worklist context user
  BPMUser bpmUser = mIdentityService.lookupUser(ctx.getUser());
  // check for BPMProcessManager role
  if (bpmUser.isInRole("BPMProcessingManager "))
     canEditTaskHeaderPriority = true;
 }
 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
```

Building a Worklist Application Using the Worklist Service APIs

The Worklist Application that is included with Oracle BPEL Process Manager provides a good starting point for building applications from scratch or for building composite applications.

The typical sequence of calls to the worklist service is as follows:

1. Get the handle to the worklist service.
2. Authenticate the user with the username and password and get a handle to the worklist context.
3. Set the required filters for retrieving tasks and get the list of tasks that qualify.
4. Loop through the task list and display tasks.
5. Perform actions on specific tasks.

A code example for calling the regular (local) worklist service APIs for performing this sequence of actions is as follows:

```
// required imports
import oracle.tip.pc.api.worklist.*;
import oracle.tip.pc.services.hw.worklist.WorklistService;
// 1. get a handle to the worklist service
WorklistService wlSvc = WorklistService.getWorklistService();

// 2. get worklist context for current user after authentication
String user = "jstein";
String password = “welcome”;
IWorklistContext ctx = wlSvc.authenticateUser(user, password);

// 3. set filters for retrieving “My” tasks with “Assigned” status
// sorted by task title is ascending order
Map filterMap = new HashMap();
filterMap.put(IWorklistService.FILTER_TYPE_TASK_FILTER,
              IWorklistService.TASK_FILTER_MY);
filterMap.put(IWorklistService.FILTER_TYPE_STATUS_FILTER,
              IWorklistService.STATUS_FILTER_ASSIGNED);
String keywords = “”;

List tasks = wlSvc.getWorklistTasks(ctx,
             keywords,
             filterMap,
             IWorklistService.SORT_FIELD_TASK_TITLE,
             IWorklistService.SORT_ORDER_ASCENDING);

// 4. APPROVE all Vacation Request tasks that are assigned to user
if (tasks != null)
{
  for (int i=0; i<tasks.size(); i++)
  {
    Task task = (Task) tasks.get(i);
    String taskId = task.getTaskId();
    // check if the task is a vacation request
    if (task.getTitle().startsWith("Vacation request"))
    {
      wlSvc.appendTaskComments(ctx, taskId, "foo");
      // now approve the task
      String action = "APPROVE";
      wlSvc.customTaskOperation(ctx, taskId, action);
    }
  }
}
```

See Also: *Oracle_ Home*\integration\orabpel\docs\workflow\index.html for Javadoc that describes the worklist service APIs

Worklist Service APIs

The following worklist service APIs are needed for various actions.

```
/*
 * Perform user authentication
 */
public IWorklistContext authenticateUser(String user,
                                         String password)
throws PCException;

/*
 * Get information for the specified group
 */
public IWorklistGroup getGroupInfo(IWorklistContext ctx,
                                   String group)
throws PCException;

/*
 * Get information for the specified user
 */
public IWorklistUser getUserInfo(IWorklistContext ctx,
                                 String user)
throws PCException;
```

```
/*
 * getWorklistTasks retrieves qualifying tasks from the start
 * row to the last row for the given filter/sort criterion
 * and user using the paging
 *
 */

public List getWorklistTasks(IWorklistContext ctx,
                             String keywords,
                             Map filterMap,
                             String sortField,
                             String sortOrder,
                             int startRow,
                             int lastRow)
throws PCException;


/*
 * getWorklistTaskDetails retrieves the details for the
 * specified task
 */
public IWorklistTask getWorklistTaskDetails(IWorklistContext ctx,
                                            String taskId)
throws PCException;

/*
 * getWorklistTaskVersionDetails retrieves the details for the
 * specified version of the task
 */
public IWorklistTask getWorklistTaskVersionDetails(
                         IWorklistContext ctx,
                         String taskId,
                         int version)
throws PCException;

/*
 * getWorklistTaskHistory retrieves the history for the
 * specified task
 */
public List getWorklistTaskHistory(IWorklistContext ctx,
                                   String taskId)
throws PCException;

/*
 * Return a list of deployed business process names
 */
public List getWorklistTaskBusinessProcesses(IWorklistContext ctx)
throws PCException;

/*
 * acquireTask lets the user acquire the task
 */
public IWorklistTask acquireTask(IWorklistContext ctx, String taskId)
throws PCException;

/*
 * releaseTask lets the user release the task
 */
public IWorklistTask releaseTask(IWorklistContext ctx, String taskId)
throws PCException;
```

```
/*
 * escalateTask lets the user escalate the task
 */
public IWorklistTask escalateTask(IWorklistContext ctx,
                                  String taskId)
throws PCException;

/*
 * renewTask lets the user renew the task
 */
public IWorklistTask renewTask(IWorklistContext ctx,
                               String taskId,
                               String durationDays)
throws PCException;

/*
 * withdrawTask lets the user withdraw the task
 */
public IWorklistTask withdrawTask(IWorklistContext ctx,
                                  String taskId)
throws PCException;

/*
 * suspendTask lets the user suspend the task
 */
public IWorklistTask suspendTask(IWorklistContext ctx, String taskId)
throws PCException;

/*
 * resumeTask lets the user resume the task
 */
public IWorklistTask resumeTask(IWorklistContext ctx, String taskId)
throws PCException;

/*
 * requestInfoForTask lets the user request additional info for the task
 */
public IWorklistTask requestInfoForTask(IWorklistContext ctx,
                                        String taskId,
                                        String user,
                                        boolean reapprovalNeeded,
                                        String comments)
throws PCException;

/*
 * submitInfoForTask lets the user submit additional info for the task
 */
public IWorklistTask submitInfoForTask(IWorklistContext ctx,
                                       String taskId)
throws PCException;

/*
 * delegateTask lets the user delegate (reassign) the task
 */
public IWorklistTask delegateTask(IWorklistContext ctx,
                                  String taskId,
                                  List worklistAssignees)
throws PCException;
```

```
/*
 * customTaskOperation lets the user perform custom operation on the task
 */
public IWorklistTask customTaskOperation(IWorklistContext ctx,
                                         String taskId,
                                         String operation,
                                         String comments)
throws PCException;

/*
 * completeAndRouteTask lets the user route the task to another user
after adding a conclusion
 */
public IWorklistTask completeAndRouteTask(IWorklistContext ctx,
                                          String taskId,
                                          String conclusion,
                                          String comments,
                                          List worklistAssignees)
throws PCException;

/*
 * updateTask lets the user update the task
 */
public void updateTask(IWorklistContext ctx, IWorklistTask task)
throws PCException;

/*
 * appendTaskComments lets the user appends comments to the task
 */
public IWorklistTask appendTaskComments(IWorklistContext ctx,
                                        String taskId,
                                        String comments)
throws PCException;

/*
 * addTaskAttachment lets the user add an attachment to the task
 */
public void addTaskAttachment(IWorklistContext ctx,
                              String taskId,
                              String name,
                              String value,
                              String type,
                              InputStream dataStream)
throws PCException;

/*
 * removeTaskAttachment lets the user delete attachments from the task
 */
public void removeTaskAttachment(IWorklistContext ctx,
                                 String taskId,
                                 String[] names)
throws PCException;

/*
 * lookupAssignees gets a list of assignee of the specified assignee type
 * (users/groups/reportees) who match the userLookupList
 */
public List lookupAssignees(IWorklistContext ctx,
                            String assigneeType,
                            String userLookupList)
```

```
throws PCException;

}
```

Example: Reassigning a Task in a Worklist Application

"Worklist Service APIs" on page 17-41 lists the API `worklistService.lookupAssignees()`, which you use to look up a user. In the following example, the lookup call searches for users that match the string `j*` and then those users are assigned to the task.

```
   List worklistAssignees = worklistService.lookupAssignees(ctx,
IWorklistService.ASSIGNEE_TYPE_USER, "j*");
   worklistService.delegateTask(ctx, taskId, worklistAssignees);
```

In this example, the `lookupAssignees` call returns `jcooper`, `jaustin`, and `jstein` (users in the demo user community that match `j*`). If you want to assign the task to `jcooper`, then use `jcooper` as the lookup string. Alternatively, you can loop through the `worklistAssignees` list that is returned and filter out entries that you do not want. The element of the list contains `ITaskAssignee` objects.

See Also:

- The `worklistxpress` sample in `*Oracle_ Home*\integration\orabpel\samples\hw`
- The worklist API available in `*Oracle_ Home*\integration\orabpel\docs\workflow\index.html`

Building a Worklist Application Using the Worklist Service Remote APIs

It is possible to build an application that works in a remote container that references the BPEL container. The worklist service supports session bean-based remote APIs. To use this approach, the JNDI and classpath configuration must be set up correctly. In the client Java program, the first step is to create a `RemoteWorklistServiceClient` instance and call its `init()` method. The `init()` method loads the JNDI properties based on the classpath and creates a local object that enables the client to talk to the remote server (the container where Oracle BPEL Process Manager is running). After a handle to `RemoteWorklistServiceClient` is obtained, most other operations are simple.

The following example shows a code fragment for using the remote service.

```
    // 1. get a handle to the remove worklist service client
    client = new RemoteWorklistServiceClient();
    client.init();

    // 2. set approver's user and password
    String user = "jstein";
    String password = globalPassword;
    System.out.println("vacationRequestTest: connecting as " + user);

    // 3. get worklist context for user
    IWorklistContext ctx = client.authenticateUser(user, password);
    System.out.println("vacationRequestTest: got Worklist Context");

    // 4. set filters for retrieving My tasks with Assigned status
    Map filterMap = new HashMap();

filterMap.put(IWorklistService.FILTER_TYPE_TASK_FILTER,
                  IWorklistService.TASK_FILTER_MY);
   filterMap.put(IWorklistService.FILTER_TYPE_STATUS_FILTER,
                 IWorklistService.STATUS_FILTER_ASSIGNED);

    List tasks = client.getWorklistTasks(ctx,
                     filterMap,
                     IWorklistService.SORT_FIELD_TASK_TITLE,
                     IWorklistService.SORT_ORDER_ASCENDING);

    // 5. APPROVE all vacationRequest tasks that are assigned to user
    if (tasks != null)
    {
      for (int i=0; i<tasks.size(); i++)
      {
        Task task = (Task) tasks.get(i);
        String taskId = task.getTaskId();
        // check if the task is a vacation request
        if (task.getTitle().startsWith("Vacation request"))
        {
          // get custom actions that can be performed
          List customActions = client.getCustomActions(ctx, taskId);
          for (int j=0; j<customActions.size(); j++)
          {
            String customAction = (String) customActions.get(j);
          }
          // get system actions that can be performed
          List systemActions = client.getSystemActions(ctx, taskId);
          for (int j=0; j<systemActions.size(); j++)
          {
            String systemAction = (String) systemActions.get(j);
          }
          // add some comments to the task
          client.appendTaskComments(ctx, taskId, "foo");
          // now approve the task
          String action = "APPROVE";
          client.customTaskOperation(ctx, taskId, action);
        }
      }
    }
```

See Also: `*Oracle_Home*\integration\orabpel\docs\workflow\index.html` for Javadoc that describes the worklist service remote APIs

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. This application is available in many languages. Instructions are provided for customizing the Worklist Application and for building your own Worklist Application using the worklist service APIs.

18 Sensors

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 contains the following topics:

- Use Cases for Sensors
- Overview of Sensor Concepts
- Implementing Sensors and Sensor Actions in JDeveloper BPEL Designer
- Sensors and Oracle BPEL Console
- 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:** `Oracle_Home\integration\orabpel\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 JDeveloper BPEL Designer 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 monitor variables of the activity also.

- 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 JDeveloper BPEL Designer, 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 18-10 and "Configuring Sensor Actions" on page 18-12 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
- List of sensors—the sensors for a sensor action

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 Console.

Use the following public views for SQL access to sensor values from any application interested in the data. These views are populated when a sensor action is used to publish to a database.

BPEL Reporting Schema

The database publisher persists the sensor data in a predefined relational schema in the database. You can use SQL to query the sensor values in the following public views from clients such as Oracle Warehouse Builder or OracleAS Portal.

Note: In Table 18–1 through Table 18–7, 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.

BPEL_ALL_PROCESSES

This view contains all the deployed BPEL processes across the Oracle BPEL Process Manager domains.

Table 18–1 BPEL_ALL PROCESSES View

Attribute Name	SQL Type	Attribute Size	Indexed or Unique?	Null?	Comment
NAME	NVARCHAR2	100	U1,1	N	Name of the BPEL process
REVISION	VARCHAR2	50	U1,2	N	Revision of the BPEL process
DOMAIN_ID	VARCHAR2	50	U1,3	N	Oracle BPEL Process Manager domain name
BASE_URL	NVARCHAR2	256	--	N	The base URL of the BPEL suite
SENSOR_URL	NVARCHAR2	256	--	N	The URL of the sensor file
SENSOR_ ACTION_URL	NVARCHAR2	256	--	N	The URL of the sensor action file

BPEL_PROCESS_ANALYSIS_REPORT

This view contains all the process instances of Oracle BPEL Process Manager.

Table 18–2 BPEL_PROCESS_ANALYSIS_REPORT View

Attribute Name	SQL Type	Attribute Size	Indexed or Unique?	Null?	Comment
ID	NUMBER	--	PK	N	Unique instance ID
BPEL_PROCESS_ NAME	NVARCHAR2	100	--	N	Name of the BPEL process
BPEL_PROCESS_ REVISION	VARCHAR2	50	--	N	Revision of the BPEL process
DOMAIN_ID	VARCHAR2	50	--	N	Oracle BPEL Process Manager domain name
TITLE	VARCHAR2	50	--	Y	User-defined title of the BPEL process
STATE	NUMBER	--	--	Y	State of the BPEL process instance
STATE_TEXT	VARCHAR2	--	--	Y	Text presentation of the state attribute
PRIORITY	NUMBER	--	--	Y	User-defined priority of the BPEL process instance
STATUS	VARCHAR2	100	--	Y	User-defined status of the BPEL process
STAGE	VARCHAR2	100	--	Y	User-defined stage property of a BPEL process
CONVERSATION_ ID	VARCHAR2	100	--	Y	User-defined conversation id of a BPEL process

Table 18–2 (Cont.) BPEL_PROCESS_ANALYSIS_REPORT View

Attribute Name	SQL Type	Attribute Size	Indexed or Unique?	Null?	Comment
CREATION_DATE	TIMESTAMP	--	--	N	The creation time stamp of the process instance
MODIFY_DATE	TIMESTAMP	--	--	Y	Time stamp when the process instance was modified
TS_DATE	DATE	--	--	Y	Date portion of `modify_date`
TS_HOUR	NUMBER	--	--	Y	Hour portion of `modify_date`
EVAL_TIME	NUMBER	--	--	Y	Time in milliseconds that a process takes to complete
SLA_ COMPLETION_ TIME	NUMBER	--	--	Y	SLA completion time in milliseconds. This is populated with the value of an optional property you can set in `bpel.xml`. For example, `<configurations>` `...` `<property name="SLACompletionTime">POYT1.5S </property>`
SLA_SATISFIED	VARCHAR2	1	--	Y	`Y` means `SLA` satisfied: `SLA_ COMPLETION_TIME < EVAL_TIME`. `N` means `SLA` not satisfied; `SLA_ COMPLETION_TIME > EVAL_TIME`. `NULL` means that no `SLACompletion` property was set.

BPEL_SENSOR_PROCESS_INSTANCES

This view is a subset of `BPEL_PROCESS_ANALYSIS_REPORT` and contains those process instances for which sensors are defined for the corresponding BPEL process and for which a sensor has fired at least once.

Table 18–3 BPEL_SENSOR_PROCESS_INSTANCES View

Attribute Name	SQL Type	Attribute Size	Indexed or Unique?	Null?	Comment
ID	NUMBER	--	PK	N	Unique instance ID
BPEL_PROCESS_ NAME	NVARCHAR2	100	U1,1	N	Name of the BPEL process
BPEL_PROCESS_ REVISION	VARCHAR2	50	U1,2	N	Revision of the BPEL process
DOMAIN_ID	VARCHAR2	50	U1,3	N	Oracle BPEL Process Manager domain name
TITLE	VARCHAR2	50	--	Y	User-defined title of the BPEL process
STATE	NUMBER	--	--	Y	State of the BPEL process instance
STATE_TEXT	VARCHAR2	--	--	Y	Text presentation of the state attribute
PRIORITY	NUMBER	--	--	Y	User-defined priority of the BPEL process instance
STATUS	VARCHAR2	100	--	Y	User-defined status of the BPEL process

Table 18–3 (Cont.) BPEL_SENSOR_PROCESS_INSTANCES View

Attribute Name	SQL Type	Attribute Size	Indexed or Unique?	Null?	Comment
STAGE	VARCHAR2	100	--	Y	User-defined stage property of a BPEL process
CONVERSATION_ ID	VARCHAR2	100	--	Y	User-defined conversation id of a BPEL process
CREATION_DATE	TIMESTAMP	--	--	N	Creation time stamp of the instance
MODIFY_DATE	TIMESTAMP	--	--	Y	Time stamp when the process instance was modified
TS_DATE	DATE	--	--	Y	Date portion of `modify_date`
TS_HOUR	NUMBER	--	--	Y	Hour portion of `modify_date`
EVAL_TIME	NUMBER	--	--	Y	Time in milliseconds that a process takes to complete
SLA_ COMPLETION_ TIME	NUMBER	--	--	Y	SLA completion time in milliseconds. This is populated with the value of an optional property you can set in `bpel.xml`. For example, `<configurations>` `...` `<property name="SLACompletionTime">POYT1.5S </property>`
SLA_SATISFIED	VARCHAR2	1	--	Y	`Y` means `SLA` satisfied: `SLA_ COMPLETION_TIME < EVAL_TIME`. `N` means `SLA` not satisfied; `SLA_ COMPLETION_TIME > EVAL_TIME`. `NULL` means that no `SLACompletion` property was set.
INSTANCE_KEY	NUMBER	--	U1	N	Corresponds to the instance key used in the Oracle BPEL Process Manager client APIs

BPEL_ACTIVITY_SENSOR_VALUES

This view contains all the activity sensor values of the monitored BPEL processes.

Table 18–4 BPEL_ACTIVITY_SENSOR_VALUES View

Attribute Name	SQL Type	Attribute Size	Indexed or Unique?	Null?	Comment
ID	NUMBER	--	PK	N	Unique ID
PROCESS_ INSTANCE	NUMBER	--	--	N	ID of process instance
BPEL_PROCESS_ NAME	NVARCHAR2	100	U1,1	N	Name of the BPEL process
BPEL_PROCESS_ REVISION	VARCHAR2	50	U1,2	N	Revision of the BPEL process
DOMAIN_ID	VARCHAR2	50	U1,3	N	Oracle BPEL Process Manager domain name
SENSOR_NAME	NVARCHAR2	100	--	N	The name of the sensor that fired

Table 18–4 (Cont.) BPEL_ACTIVITY_SENSOR_VALUES View

Attribute Name	SQL Type	Attribute Size	Indexed or Unique?	Null?	Comment
`SENSOR_TARGET`	`NVARCHAR2`	256	--	N	The target of the fired sensor
`ACTION_NAME`	`NVARCHAR2`	100	--	N	The name of the sensor action
`ACTION_FILTER`	`NVARCHAR2`	256	--	Y	The filter of the action
`CREATION_DATE`	`TIMESTAMP`	--	--	N	The creation date of the activity sensor value
`MODIFY_DATE`	`TIMESTAMP`	--	--	Y	The time stamp of last modification
`TS_DATE`	`DATE`	--	--	Y	Date portion of `modify_date`
`TS_HOUR`	`NUMBER`	--	--	Y	Hour portion of `modify_date`
`CRITERIA_ SATISFIED`	`VARCHAR2`	1	--	Y	`NULL`, `Y`, or `N`
`ACTIVITY_NAME`	`NVARCHAR2`	100	--	N	The name of the BPEL activity
`ACTIVITY_TYPE`	`VARCHAR2`	30	--	N	The type of the BPEL activity
`ACTIVITY_ STATE`	`VARCHAR2`	30	--	Y	The state of the activity
`EVAL_POINT`	`VARCHAR2`	20	--	N	The evaluation point of the activity sensor
`ERROR_MESSAGE`	`NVARCHAR2`	2000	--	Y	An error message
`RETRY_COUNT`	`NUMBER`	--	--	Y	The number of retries of the activity
`EVAL_TIME`	`NUMBER`	--	--	Y	Time in milliseconds that an activity takes to complete
`INSTANCE_KEY`	`NUMBER`	--	NU1	N	Corresponds to the instance key used in the Oracle BPEL Process Manager client APIs

BPEL_FAULT_SENSOR_VALUES

This view contains all the fault sensor values.

Table 18–5 BPEL_FAULT_SENSOR_VALUES View

Attribute Name	SQL Type	Attribute Size	Indexed or Unique?	Null?	Comment
`ID`	`NUMBER`	--	PK	N	Unique ID
`PROCESS_ INSTANCE`	`NUMBER`	--	--	N	ID of process instance
`BPEL_PROCESS_ NAME`	`NVARCHAR2`	100	U1,1	N	Name of the BPEL process
`BPEL_PROCESS_ REVISION`	`VARCHAR2`	50	U1,2	N	Revision of the BPEL process
`DOMAIN_ID`	`VARCHAR2`	50	U1,3	N	Oracle BPEL Process Manager domain name
`SENSOR_NAME`	`NVARCHAR2`	100	--	N	The name of the sensor that fired
`SENSOR_TARGET`	`NVARCHAR2`	256	--	N	The target of the fired sensor
`ACTION_NAME`	`NVARCHAR2`	100	--	N	The name of the sensor action

Table 18–5 (Cont.) BPEL_FAULT_SENSOR_VALUES View

Attribute Name	SQL Type	Attribute Size	Indexed or Unique?	Null?	Comment
ACTION_FILTER	NVARCHAR2	256	--	Y	The filter of the action
CREATION_DATE	TIMESTAMP	--	--	N	The creation date of the activity sensor value
MODIFY_DATE	TIMESTAMP	--	--	Y	The time stamp of last modification
TS_DATE	DATE	--	--	Y	Date portion of `modify_date`
TS_HOUR	NUMBER	--	--	Y	Hour portion of `modify_date`
CRITERIA_ SATISFIED	VARCHAR2	1	--	Y	`NULL` if no action filter specified, `Y` if action filter specified and evaluates to true, `N` otherwise
ACTIVITY_NAME	NVARCHAR2	100	--	N	The name of the BPEL activity
ACTIVITY_TYPE	VARCHAR2	30	--	N	The type of the BPEL activity
MESSAGE	CLOB	--	--	Y	The fault message
INSTANCE_KEY	NUMBER	--	NU1	N	Corresponds to the instance key used in the Oracle BPEL Process Manager client APIs

BPEL_VARIABLE_SENSOR_VALUES

This view contains all the variable sensor values.

Table 18–6 BPEL_VARIABLE_SENSOR_VALUES View

Attribute Name	SQL Type	Attribute Size	Indexed or Unique?	Null?	Comment
ID	NUMBER	--	PK	N	Unique ID
PROCESS_ INSTANCE	NUMBER	--	--	N	ID of process instance
BPEL_PROCESS_ NAME	NVARCHAR2	100	U1,1	N	Name of the BPEL process
BPEL_PROCESS_ REVISION	VARCHAR2	50	U1,2	N	Revision of the BPEL process
DOMAIN_ID	VARCHAR2	50	U1,3	N	Oracle BPEL Process Manager domain name
SENSOR_NAME	NVARCHAR2	100	--	N	Name of the sensor that fired
SENSOR_TARGET	NVARCHAR2	256	--	N	Target of the sensor
ACTION_NAME	NVARCHAR2	100	--	N	Name of the action
ACTION_FILTER	NVARCHAR2	256	--	Y	Filter of the action
ACTIVITY_ SENSOR	NUMBER	--	--	Y	ID of corresponding activity sensor value
CREATION_DATE	TIMESTAMP	--	--	N	Creation date
TS_DATE	DATE	--	--	N	Date portion of `creation_date`
TS_HOUR	NUMBER	--	--	N	Hour portion of `creation_date`
VARIABLE_NAME	NVARCHAR2	256	--	N	The name of the BPEL variable

Table 18–6 (Cont.) BPEL_VARIABLE_SENSOR_VALUES View

Attribute Name	SQL Type	Attribute Size	Indexed or Unique?	Null?	Comment
CRITERIA_ SATISFIED	VARCHAR2	1	--	Y	NULL, Y, or N
TARGET	NVARCHAR2	256	--	--	--
UPDATER_NAME	NVARCHAR2	100	--	N	The name of the activity or event that updated the variable
UPDATER_TYPE	NVARCHAR2	100	--	N	The type of the BPEL activity or event
VALUE_TYPE	SMALLINT	--	--	N	The value type of the variable (corresponds to `java.sql.Types` values)
VARCHAR2_ VALUE	NVARCHAR2	2000	--	Y	The value of string-like variables
NUMBER_VALUE	NUMBER	--	--	Y	The value of number-like variables (such as `float`, `double`, and `int`)
DATE_VALUE	TIMESTAMP	--	--	Y	The value of date-like variables
BLOB_VALUE	BLOB	--	--	Y	The value of binary data variables
CLOB_VALUE	CLOB	--	--	Y	The value of CLOB-like variables (XML)
INSTANCE_KEY	NUMBER	--	NU1	N	Corresponds to the instance key used in the Oracle BPEL Process Manager client APIs

BPEL_ERRORS

This view gives an overview of all the errors from BPEL services.

Table 18–7 BPEL_ERRORS View

Attribute Name	SQL Type	Attribute Size	Indexed or Unique?	Null?	Comment
ID	NUMBER	--	PK	N	Unique ID
BPEL_PROCESS_ NAME	NVARCHAR2	100	U1,1	N	Name of the BPEL process
BPEL_PROCESS_ REVISION	VARCHAR2	50	U1,2	N	Revision of the BPEL process
DOMAIN_ID	VARCHAR2	50	U1,3	N	Oracle BPEL Process Manager domain name
CREATION_DATE	TIMESTAMP	--	--	N	The creation date of the activity sensor value
TS_DATE	DATE	--	--	N	Date portion of `creation_date`
TS_HOUR	NUMBER	--	--	N	Hour portion of `creation_date`
ERROR_CODE	NUMBER	--	--	N	Error code
EXCEPTION_ TYPE	NUMBER	--	--	N	Type of the error
EXCEPTION_ SEVERITY	NUMBER	--	--	N	Severity of the error

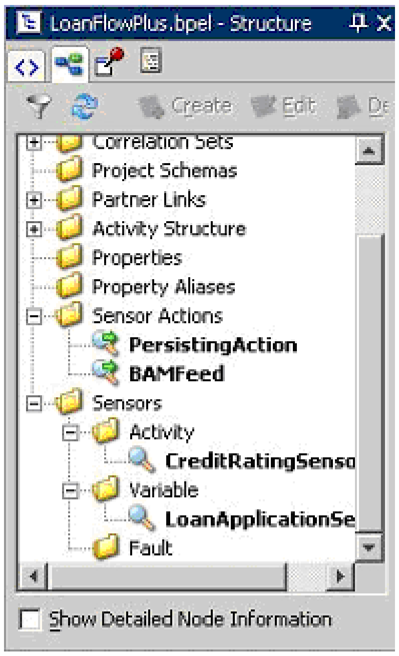
Table 18–7 (Cont.) BPEL_ERRORS View

Attribute Name	SQL Type	Attribute Size	Indexed or Unique?	Null?	Comment
EXCEPTION_ NAME	NVARCHAR2	200	--	N	Name of the error
EXCEPTION_ DESCRIPTION	NVARCHAR2	2000	--	Y	A short description of the error
EXCEPTION_FIX	NVARCHAR2	2000	--	Y	A description on how to fix the error
EXCEPTION_ CONTEXT	VARCHAR2	4000	--	Y	The context of the error
COMPONENT	NUMBER	--	--	N	The BPEL component that caused the error
THREAD_ID	VARCHAR2	200	--	N	The Java thread name where the error occurred
STACKTRACE	CLOB	--	--	N	The Java stack trace

Implementing Sensors and Sensor Actions in JDeveloper BPEL Designer

In JDeveloper BPEL Designer, sensors and sensor actions are displayed as part of the process tree structure, as shown in Figure 18–1.

Figure 18–1 Sensors and Sensor Actions Displayed in JDeveloper BPEL Designer



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**.

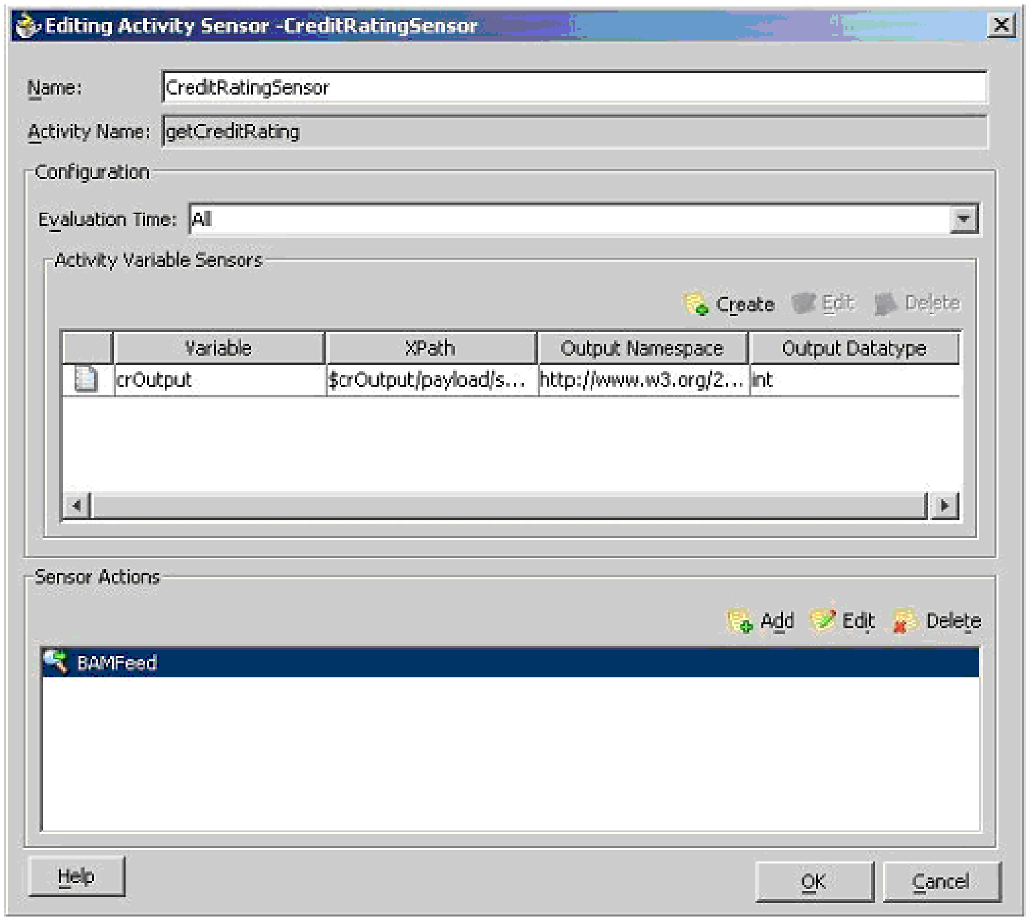
Using `LoanDemoPlus` as an example, the following sections describe how to configure sensors and sensor actions.

See Also: The `LoanDemoPlus` tutorial, at `*Oracle_Home*\integration\orabpel\samples\demos`

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 JDeveloper BPEL Designer, as shown in Figure 18–2. Activities that have sensors associated with them are indicated with a magnifying glass.

Figure 18–2 Creating an Activity Sensor



Editing Activity Sensor -CreditRatingSensor

Name: CreditRatingSensor

Activity Name: getCreditRating

Configuration

Evaluation Time: All

Activity Variable Sensors

	Variable	XPath	Output Namespace	Output Datatype
	crOutput	$crOutput/payload/s...	http://www.w3.org/2...	int

Sensor Actions

BAMFeed

The **Evaluation Time** attribute shown in Figure 18–2 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.

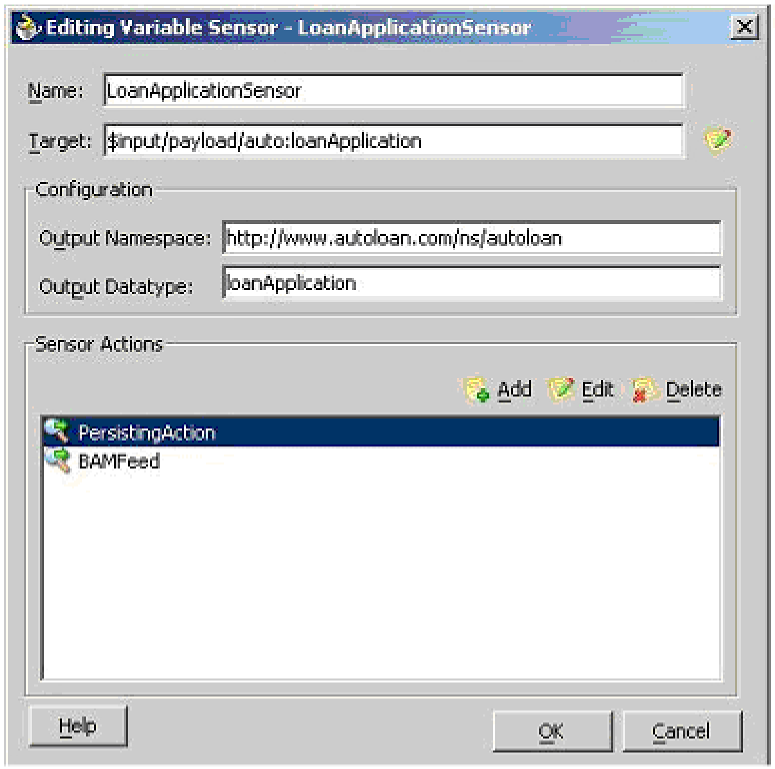
- **All**—Monitoring occurs during all of the preceding phases.

A new entry is created in the `sensor.xml` file, as follows:

```
<sensor sensorName="CreditRatingSensor"

classname="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 18–3.

Figure 18–3 Creating a Variable Sensor



Editing Variable Sensor - LoanApplicationSensor

Name: LoanApplicationSensor

Target: $input/payload/auto:loanApplication

Configuration

Output Namespace: http://www.autoloan.com/ns/autoloan

Output Datatype: loanApplication

Sensor Actions

PersistingAction

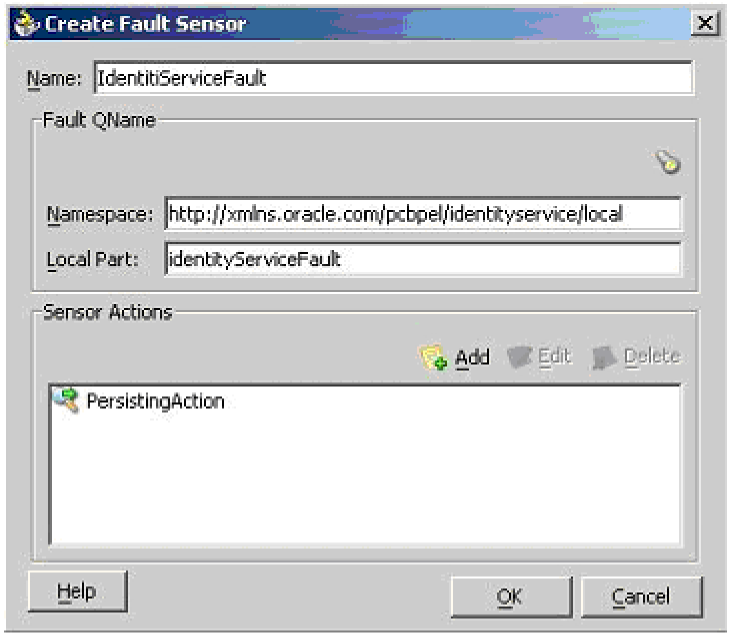
BAMFeed

A new entry is created in the `sensor.xml` file, as follows:

```
<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 18–4.

Figure 18–4 Creating a Fault Sensor



Name: IdentitiServiceFault

Namespace: http://xmlns.oracle.com/pcbpel/identityservice/local

Local Part: identityServiceFault

PersistingAction

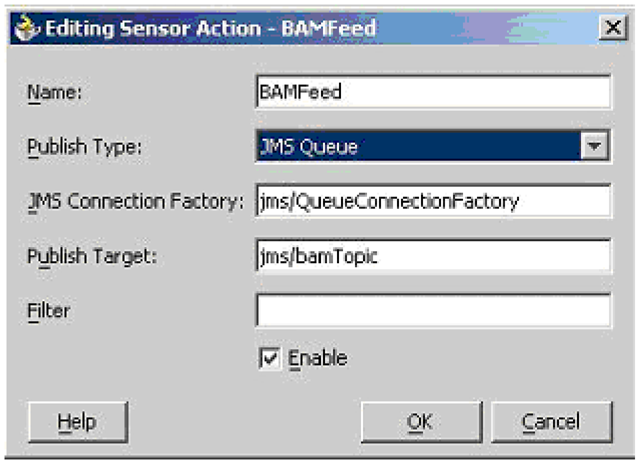
A new entry is created in the `sensor.xml` file, as follows:

```
<sensor sensorName="IdentityServiceFault"
        classname="oracle.tip.pc.services.reports.dca.agents.BpelFaultSensorAgent"
        kind="fault"
        target="is:identityServiceFault">
    <faultConfig/>
</sensor>
```

Configuring Sensor Actions

When you create sensors, you identify the activities, variables, and faults you want to monitor during run time. 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 you create a sensor action, as shown in Figure 18–5, and associate it with the `LoanApplicationSensor`.

Figure 18–5 Creating a Sensor Action



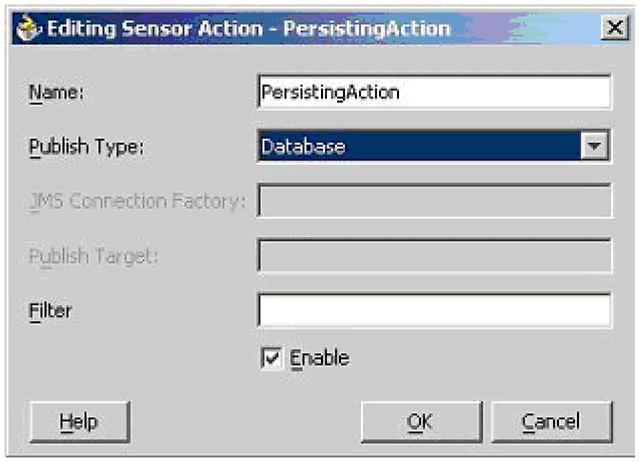
Name: BAMFeed
Publish Type: JMS Queue
JMS Connection Factory: jms/QueueConnectionFactory
Publish Target: jms/bamTopic
Filter
[✓] Enable

A new entry is created in the `sensorAction.xml` file, as follows:

```
<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 18–6, and associate it with both `CreditRatingSensor` and `LoanApplicationSensor`.

Figure 18–6 Creating an Additional Sensor Action



Name: PersistingAction
Publish Type: Database
JMS Connection Factory:
Publish Target:
Filter
[✓] Enable

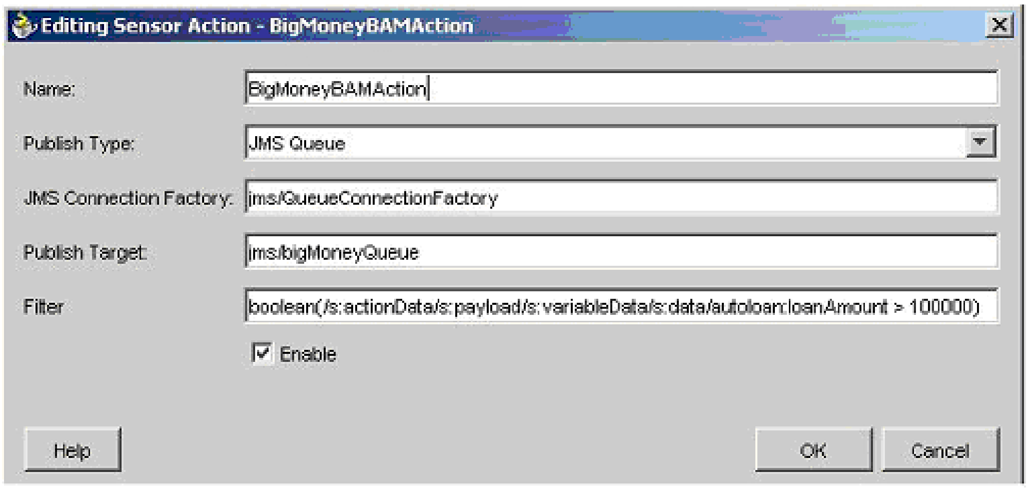
A new entry is created in the `sensorAction.xml` file, as follows:

```
<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 as well as 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 18–7.

Figure 18–7 Creating a Sensor Action with a Filter



Editing Sensor Action - BigMoneyBAMAction

Name:	BigMoneyBAMAction
Publish Type:	JMS Queue
JMS Connection Factory:	jms/QueueConnectionFactory
Publish Target:	jms/bigMoneyQueue
Filter	boolean(/s:actionData/s:payload/s:variableData/s:data/autoloan:loanAmount > 100000)
	[✓ Enable]

A new entry is created in the `sensorAction.xml` file, as follows:

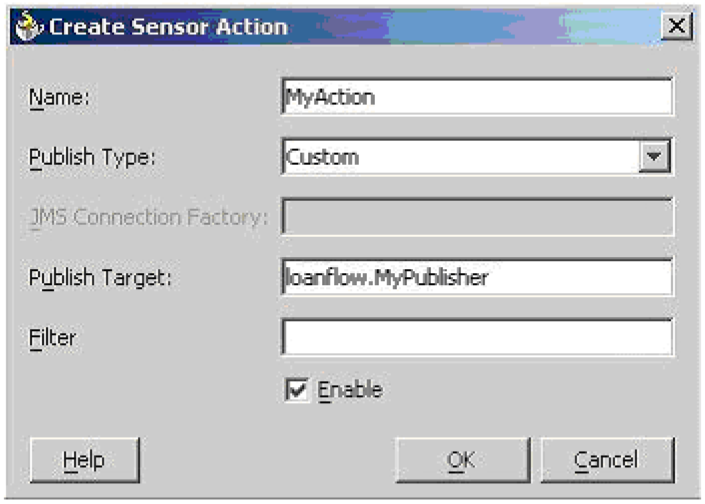
```
<action name="BigMoneyBAMAction"
        enabled='true'
        filter="boolean(/s:actionData/s:payload
                        /s:variableData/s:data
                        /autoloan:loanAmount > 100000)"
        publishType="JMSQueue"
        publishTarget="jms/bigMoneyQueue">
  <sensorName>LoanApplicationSensor</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, and JMS topic, then you can create a sensor action with the custom publish type, as shown in Figure 18–8. The name in the **Publish Target** field denotes a fully qualified Java class name that must be implemented.

Figure 18–8 Using the Custom Publish Type


Create Sensor Action

Name: MyAction

Publish Type: Custom

JMS Connection Factory:

Publish Target: loanflow.MyPublisher

Filter

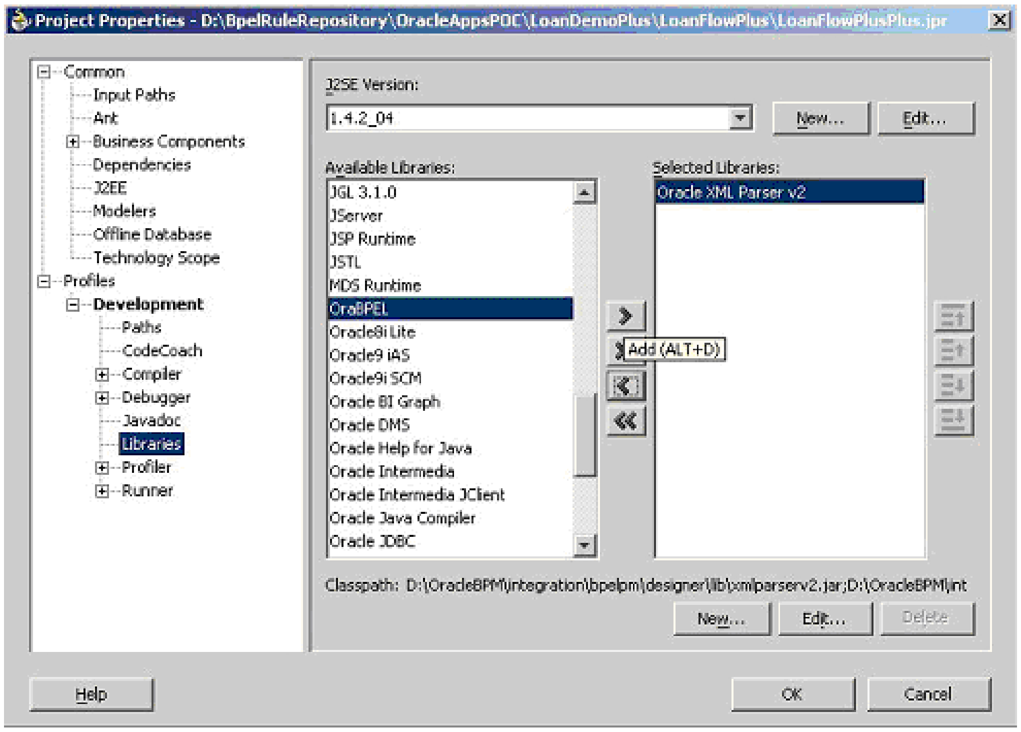
☑ Enable

Creating a Custom Data Publisher

To create a custom data publisher, double-click your BPEL project in JDeveloper BPEL Designer and do the following:

1. Open **Project Properties**, from **Libraries**, select **OraBPEL**, and add it to the **Selected Libraries** list.

   This adds the required BPEL libraries to your BPEL project.


Project Properties - D:\BpelRuleRepository\OracleAppsPOC\LoanDemoPlus\LoanFlowPlus\LoanFlowPlusPlus.jpr

- Common
  - Input Paths
  - Ant
  - Business Components
  - Dependencies
  - J2EE
  - Modelers
  - Offline Database
  - Technology Scope
- Profiles
  - **Development**
    - Paths
    - CodeCoach
    - Compiler
    - Debugger
    - Javadoc
    - Libraries
    - Profiler
    - Runner

J2SE Version: 1.4.2_04

Available Libraries:
JGL 3.1.0
JServer
JSP Runtime
JSTL
MDS Runtime
OraBPEL
Oracle8i Lite
Oracle9 iAS
Oracle9i SCM
Oracle BI Graph
Oracle DMS
Oracle Help for Java
Oracle Intermedia
Oracle Intermedia JClient
Oracle Java Compiler
Oracle JDBC

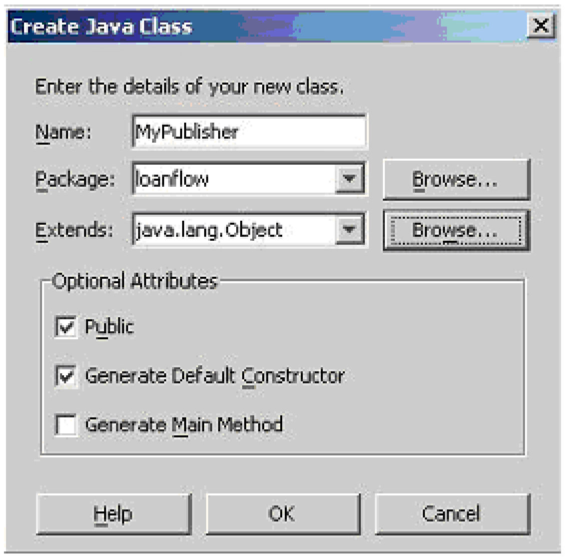
Selected Libraries:
Oracle XML Parser v2

Add (ALT+D)

Classpath: D:\OracleBPM\integration\bpelpm\designer\lib\xmlparserv2.jar;D:\OracleBPM\int

2. Create a new Java class.

   The package and class name must match the publish target name of the sensor action.



Enter the details of your new class.

Name: MyPublisher

Package: loanflow

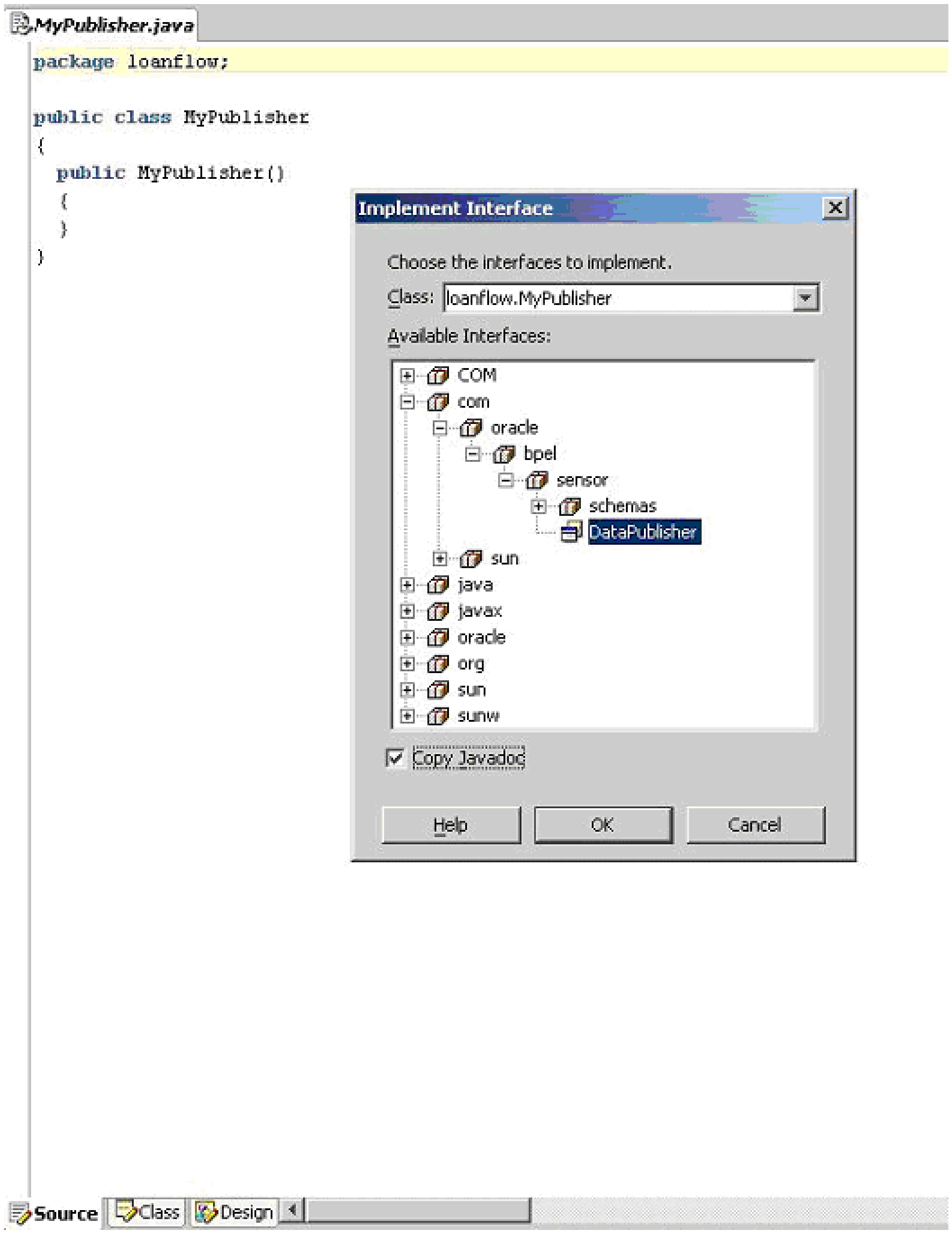
Extends: java.lang.Object

Optional Attributes

- [✓] Public
- [✓] Generate Default Constructor
- [ ] Generate Main Method

3. From **JDeveloper Tools** > **Implement Interface** > **Available Interfaces**, click **DataPublisher**.

   This updates the source file and fills in the methods and import statements of the **DataPublisher** interface.



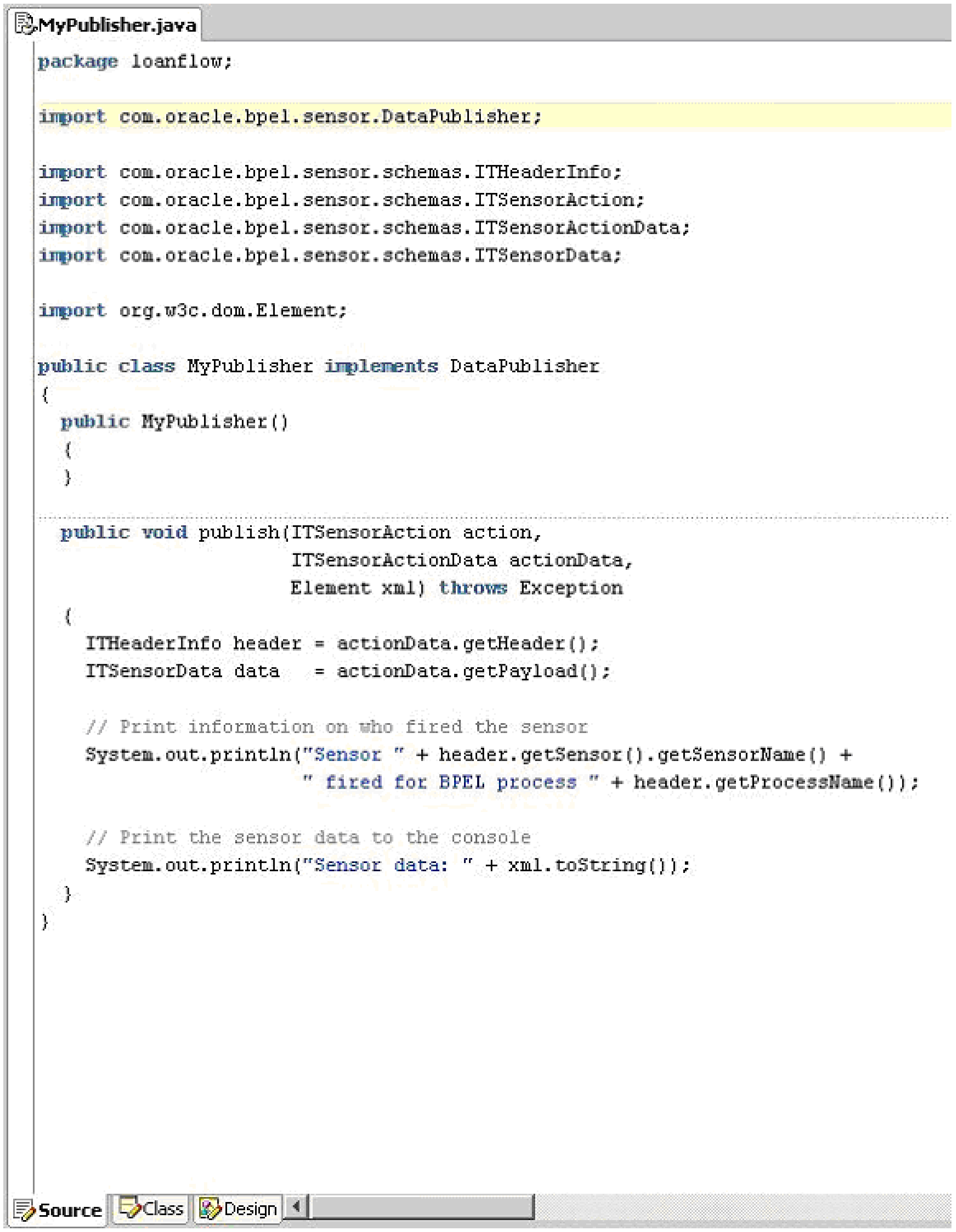
```
package loanflow;

public class MyPublisher
{
  public MyPublisher()
  {
  }
}
```

Choose the interfaces to implement.

Class: loanflow.MyPublisher

Copy Javadoc [✓]

4. Using the JDeveloper BPEL Designer editor, implement the publish method of the `DataPublisher` interface, as shown in the following sample custom data publisher class.



```
package loanflow;

import com.oracle.bpel.sensor.DataPublisher;

import com.oracle.bpel.sensor.schemas.ITHeaderInfo;
import com.oracle.bpel.sensor.schemas.ITSensorAction;
import com.oracle.bpel.sensor.schemas.ITSensorActionData;
import com.oracle.bpel.sensor.schemas.ITSensorData;

import org.w3c.dom.Element;

public class MyPublisher implements DataPublisher
{
  public MyPublisher()
  {
  }

  public void publish(ITSensorAction action,
                      ITSensorActionData actionData,
                      Element xml) throws Exception
  {
    ITHeaderInfo header = actionData.getHeader();
    ITSensorData data   = actionData.getPayload();

    // Print information on who fired the sensor
    System.out.println("Sensor " + header.getSensor().getSensorName() +
                       " fired for BPEL process " + header.getProcessName());

    // Print the sensor data to the console
    System.out.println("Sensor data: " + xml.toString());
  }
}
```

5. 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.

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.

Registering the Sensors and Sensor Actions in bpel.xml

JDeveloper BPEL Designer automatically updates the process deployment file `bpel.xml` to include appropriate properties for sensors and sensor actions, as follows:

```
<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 Console

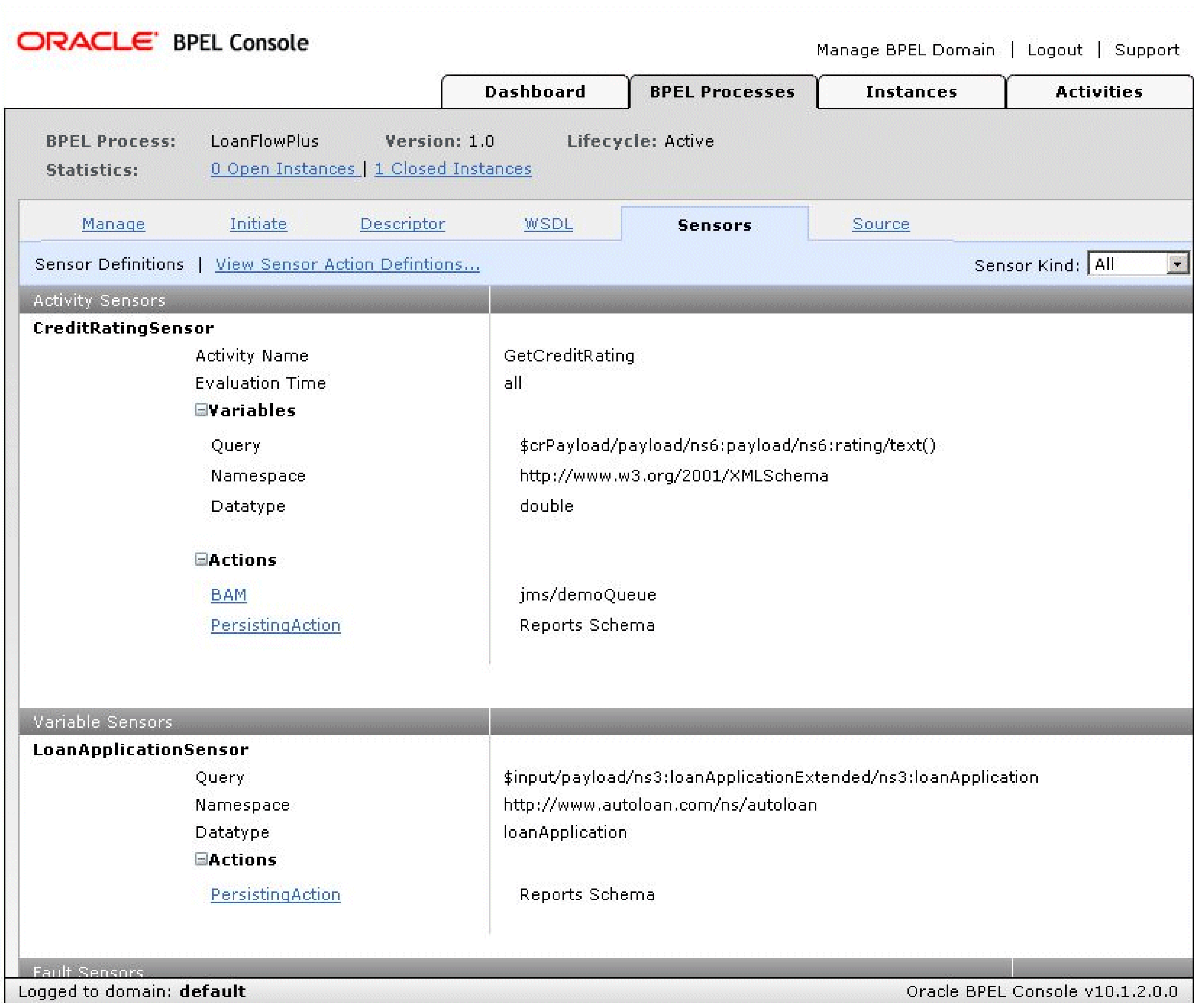
The console provides support to view the metadata of sensors and sensor actions as well as the sensor data created as part of the process execution. Access Oracle BPEL Console using Internet Explorer at

`http://`*`localhost`*`:`*`portnumber`*`/BPELConsole`

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 JDeveloper BPEL Designer. In Oracle BPEL Console, 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 18–9 is displayed.

Figure 18–9 Sensor Data on the BPEL Processes Tab of Oracle BPEL Console



BPEL Process: LoanFlowPlus Version: 1.0 Lifecycle: Active
Statistics: 0 Open Instances | 1 Closed Instances

Sensor Definitions | View Sensor Action Defintions... Sensor Kind: All

Activity Sensors

CreditRatingSensor	
Activity Name	GetCreditRating
Evaluation Time	all
Variables	
Query	$crPayload/payload/ns6:payload/ns6:rating/text()
Namespace	http://www.w3.org/2001/XMLSchema
Datatype	double
Actions	
BAM	jms/demoQueue
PersistingAction	Reports Schema

Variable Sensors

LoanApplicationSensor	
Query	$input/payload/ns3:loanApplicationExtended/ns3:loanApplication
Namespace	http://www.autoloan.com/ns/autoloan
Datatype	loanApplication
Actions	
PersistingAction	Reports Schema

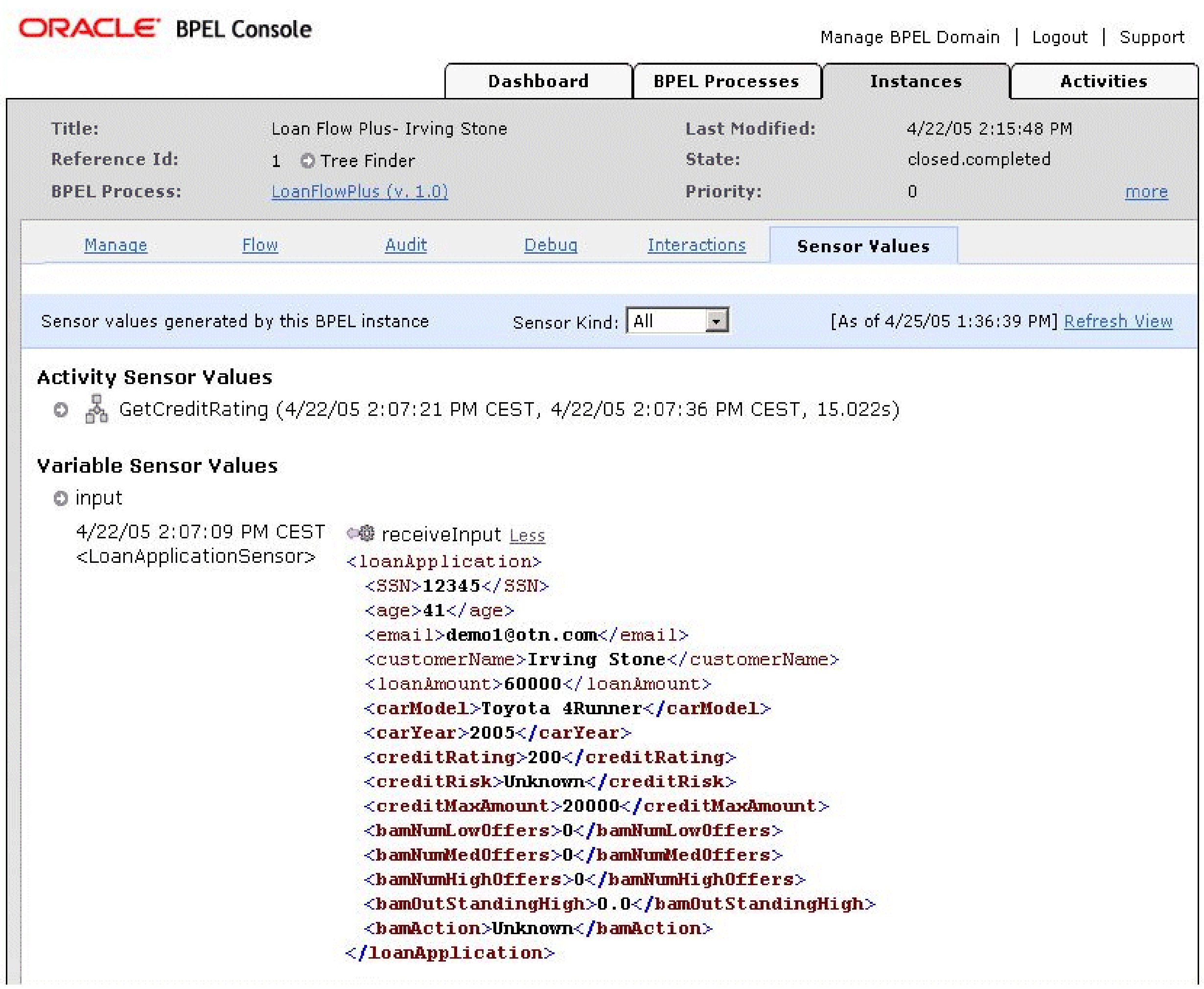
Fault Sensors

Logged to domain: **default** Oracle BPEL Console v10.1.2.0.0

Viewing Sensor Data

The console provides support to monitor sensors for which a **BpelReportsSchema** sensor action is defined. In Oracle BPEL Console, 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 18–10 is displayed.

Figure 18–10 Sensor Data on the Instances Tab of Oracle BPEL Console



Title:	Loan Flow Plus- Irving Stone	Last Modified:	4/22/05 2:15:48 PM
Reference Id:	1 Tree Finder	State:	closed.completed
BPEL Process:	LoanFlowPlus (v. 1.0)	Priority:	0

Sensor values generated by this BPEL instance Sensor Kind: All [As of 4/25/05 1:36:39 PM] Refresh View

Activity Sensor Values

GetCreditRating (4/22/05 2:07:21 PM CEST, 4/22/05 2:07:36 PM CEST, 15.022s)

Variable Sensor Values

input

4/22/05 2:07:09 PM CEST <LoanApplicationSensor> receiveInput Less

```
<loanApplication>
  <SSN>12345</SSN>
  <age>41</age>
  <email>demo1@otn.com</email>
  <customerName>Irving Stone</customerName>
  <loanAmount>60000</loanAmount>
  <carModel>Toyota 4Runner</carModel>
  <carYear>2005</carYear>
  <creditRating>200</creditRating>
  <creditRisk>Unknown</creditRisk>
  <creditMaxAmount>20000</creditMaxAmount>
  <bamNumLowOffers>0</bamNumLowOffers>
  <bamNumMedOffers>0</bamNumMedOffers>
  <bamNumHighOffers>0</bamNumHighOffers>
  <bamOutStandingHigh>0.0</bamOutStandingHigh>
  <bamAction>Unknown</bamAction>
</loanApplication>
```

Note: Only sensors associated with a database sensor action are displayed in the console. Sensors associated with a JMS queue, JMS topic, or custom sensor action are not displayed.

Summary

This chapter describes how to set up and use sensors to monitor BPEL activities, variables, and faults during run time.

Part IV

Development Life Cycle

This part describes how to run BPEL processes from Oracle BPEL Console.

This part contains the following chapters:

- Chapter 19, "BPEL Process Deployment and Domain Management"

19

BPEL Process Deployment and Domain Management

This chapter provides an overview of key BPEL process deployment and domain management concepts. An overview of Oracle BPEL Console 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
- Viewing BPEL Processes in Oracle BPEL Console
- Build and Command Line Tools
- Summary

See Also: The following documentation for tutorials in which you deploy BPEL processes:

- "Compiling and Deploying the BPEL Process" on page 3-16
- *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 Console.

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 Console to view any currently running BPEL processes before compiling and deploying additional processes.

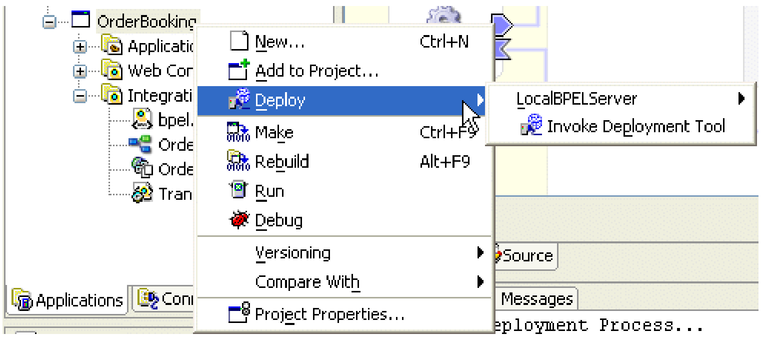
BPEL processes can be compiled and deployed in two types of BPEL designer environments:

- Compiling and Deploying on JDeveloper BPEL Designer
- Compiling and Deploying on Eclipse BPEL Designer

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 on JDeveloper BPEL Designer

To compile and deploy a BPEL process on JDeveloper BPEL Designer, right-click the BPEL project (for this example, named **OrderBooking**) and select **Deploy** in the **Applications Navigator** section:



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 Oracle BPEL Server connection automatically created during Oracle BPEL Process Manager installation (named **LocalBPELServer**). You can create additional server connections by performing the following steps:

  1. Select **Connection Navigator** from the **View** main menu in JDeveloper BPEL Designer.
  2. Right-click **BPEL Process Manager Server**.
  3. Select **New BPEL Process Manager Connection**.

This starts the BPEL Process Manager Connection wizard. After completion, this new connection appears when you right-click the process and select **Deploy**.

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.

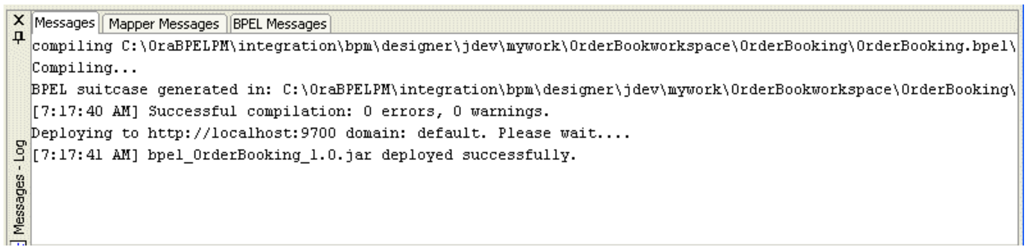
If the server mode is development, you are not prompted and the version is automatically overwritten.

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 **Applications Navigator** and select **Deploy** > ***connection_ name*** > **Deploy to default domain**. Provide a more descriptive name when prompted in the **Your Version** field of the Deploy Properties window (for example, sales_div_1.0). You can then retire the other process version on Oracle BPEL Console.

- If you select **Invoke Deployment Tool**, the Deploy Properties window opens. This window enables you to customize your settings by selecting a different or creating a new Oracle BPEL Console 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.

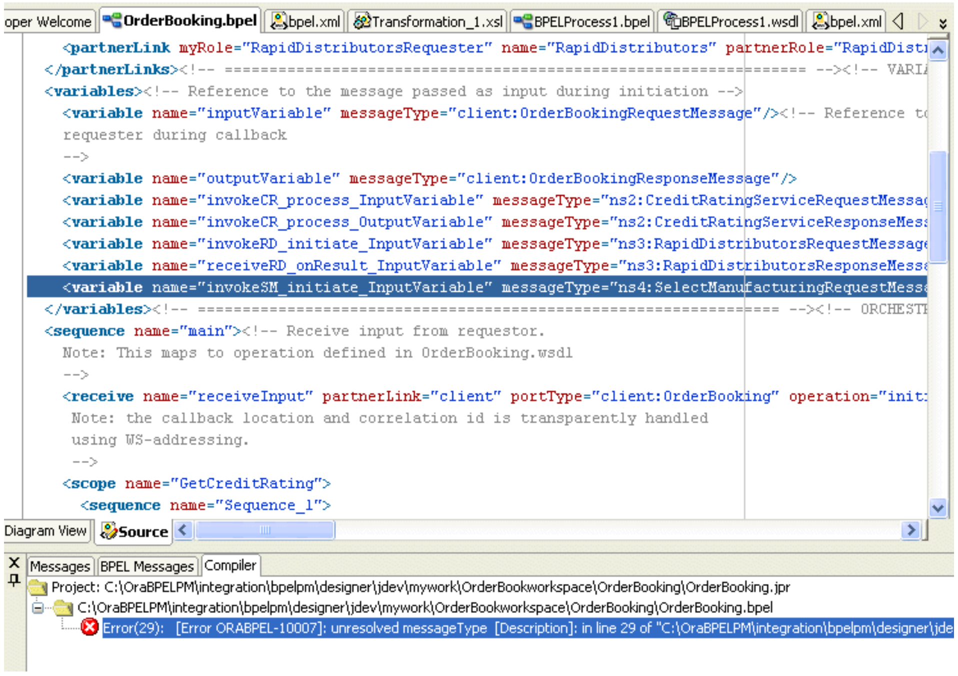
After you select a deployment method, the **Log Window** at the bottom of JDeveloper BPEL Designer displays messages about the status of the deployment. For example, the following message indicates that deployment was successful.



```
compiling C:\OraBPELPM\integration\bpm\designer\jdev\mywork\OrderBookworkspace\OrderBooking\OrderBooking.bpel\
Compiling...
BPEL suitcase generated in: C:\OraBPELPM\integration\bpm\designer\jdev\mywork\OrderBookworkspace\OrderBooking\
[7:17:40 AM] Successful compilation: 0 errors, 0 warnings.
Deploying to http://localhost:9700 domain: default. Please wait....
[7:17:41 AM] bpel_OrderBooking_1.0.jar deployed successfully.
```

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.



```
    <partnerLink myRole="RapidDistributorsRequester" name="RapidDistributors" partnerRole="RapidDistr
  </partnerLinks><!-- ================================================================= --><!-- VARIA
  <variables><!-- Reference to the message passed as input during initiation -->
    <variable name="inputVariable" messageType="client:OrderBookingRequestMessage"/><!-- Reference to
    requester during callback
    -->
    <variable name="outputVariable" messageType="client:OrderBookingResponseMessage"/>
    <variable name="invokeCR_process_InputVariable" messageType="ns2:CreditRatingServiceRequestMessag
    <variable name="invokeCR_process_OutputVariable" messageType="ns2:CreditRatingServiceResponseMess
    <variable name="invokeRD_initiate_InputVariable" messageType="ns3:RapidDistributorsRequestMessage
    <variable name="receiveRD_onResult_InputVariable" messageType="ns3:RapidDistributorsResponseMess
    <variable name="invokeSM_initiate_InputVariable" messageType="ns4:SelectManufacturingRequestMess
  </variables><!-- ================================================================= --><!-- ORCHESTR
  <sequence name="main"><!-- Receive input from requestor.
    Note: This maps to operation defined in OrderBooking.wsdl
    -->
    <receive name="receiveInput" partnerLink="client" portType="client:OrderBooking" operation="init:
     Note: the callback location and correlation id is transparently handled
     using WS-addressing.
     -->
    <scope name="GetCreditRating">
      <sequence name="Sequence_1">
```

```
Project: C:\OraBPELPM\integration\bpelpm\designer\jdev\mywork\OrderBookworkspace\OrderBooking\OrderBooking.jpr
  C:\OraBPELPM\integration\bpelpm\designer\jdev\mywork\OrderBookworkspace\OrderBooking\OrderBooking.bpel
    Error(29):  [Error ORABPEL-10007]: unresolved messageType  [Description]: in line 29 of "C:\OraBPELPM\integration\bpelpm\designer\jde
```

Make corrections and redeploy.

See Also:

- "Creating a BPEL Domain" on page 19-9
- "Changing Oracle BPEL Server Mode" on page 19-9

Compiling Without Deploying on JDeveloper BPEL Designer

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:

```
Oracle_Home\integration\jdev\jdev\default\mywork\workspace_name\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`)

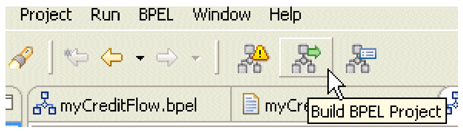
   ```
   Oracle_Home\integration\orabpel\domains\default\deploy
   ```

or

1. Log into Oracle BPEL Console using Internet Explorer 6.0 by selecting **Start** > **All Programs** > **Oracle -** ***Oracle_Home*** > **Oracle BPEL Process Manager 10.1.2** > **BPEL Console**.
2. Click **BPEL Processes**.
3. Click **Deploy New Process** in the **Related Tasks** section.
4. Click **Browse** to select the process.
5. Click **Deploy**.
6. Click the **Dashboard** tab to view the newly deployed process.

Compiling and Deploying on Eclipse BPEL Designer

To compile and deploy a BPEL process on Eclipse BPEL Designer, you click the **Build BPEL Project** icon in the tool bar:

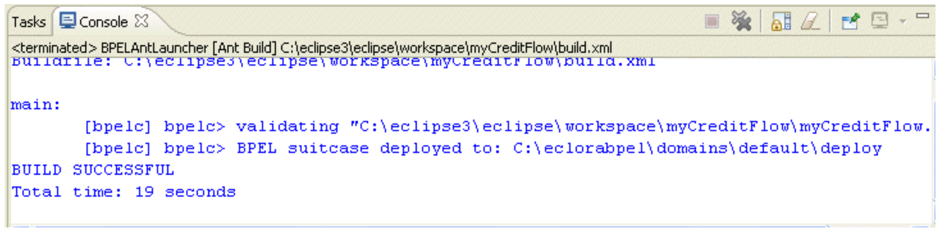


If you want to change the domain to which to deploy the process or the version label of the process, you must edit properties in the project's `build.xml` file. In this example, the domain is named `default` and the label version is `1.0`.

```
     Name of the domain the generated BPEL suitcase will be deployed to
 ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<property name="deploy" value="default"/>
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
     What version number should be used to tag the generated BPEL archive?
     ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
    <property name="rev" value="1.0"/>
```

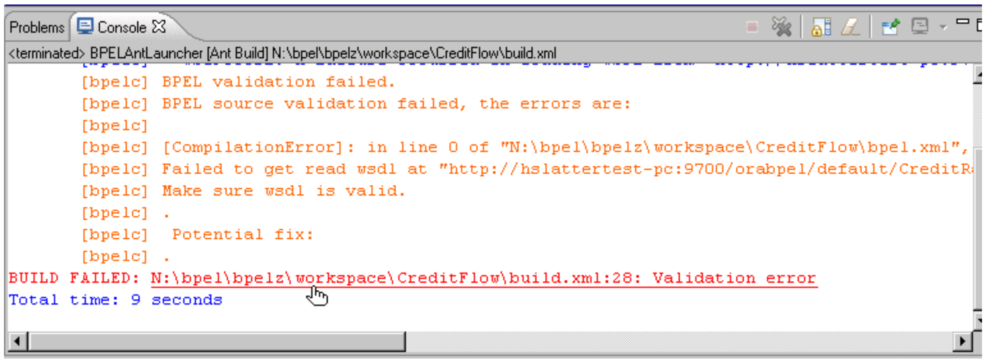
As with JDeveloper BPEL Designer, if you overwrite the version, the old version is made stale, and the new version is made active. You cannot revert to the stale definition. In addition, the stale version cannot be examined, and all flow and audit information is lost. If you enter a different version label in the `build.xml` file (for example, `2.0`), it is deployed to Oracle BPEL Server, while the older, deployed version (`1.0`) continues to run simultaneously on Oracle BPEL Server.

After you deploy the BPEL process, the **Log Window** at the bottom of Eclipse BPEL Designer displays messages about the status of the deployment. For example, the following message indicates that deployment was successful.



```
<terminated> BPELAntLauncher [Ant Build] C:\eclipse3\eclipse\workspace\myCreditFlow\build.xml
Buildfile: C:\eclipse3\eclipse\workspace\myCreditFlow\build.xml

main:
      [bpelc] bpelc> validating "C:\eclipse3\eclipse\workspace\myCreditFlow\myCreditFlow.
      [bpelc] bpelc> BPEL suitcase deployed to: C:\eclorabpel\domains\default\deploy
BUILD SUCCESSFUL
Total time: 19 seconds
```

If deployment is unsuccessful, errors display at the bottom of the window. Click the error next to the `BUILD FAILED` message to display the line of code that caused deployment to fail.



```
<terminated> BPELAntLauncher [Ant Build] N:\bpel\bpelz\workspace\CreditFlow\build.xml
       [bpelc] BPEL validation failed.
       [bpelc] BPEL source validation failed, the errors are:
       [bpelc]
       [bpelc] [CompilationError]: in line 0 of "N:\bpel\bpelz\workspace\CreditFlow\bpel.xml",
       [bpelc] Failed to get read wsdl at "http://hslattertest-pc:9700/orabpel/default/CreditR
       [bpelc] Make sure wsdl is valid.
       [bpelc] .
       [bpelc]  Potential fix:
       [bpelc] .
BUILD FAILED: N:\bpel\bpelz\workspace\CreditFlow\build.xml:28: Validation error
Total time: 9 seconds
```

Make corrections and redeploy.

See Also: "Creating and Managing a BPEL Domain" on page 19-8

BPEL Suitcase JAR File

You may have noticed the words `BPEL suitcase` that appeared in the successful deployment **Log Window** messages shown for both JDeveloper BPEL Designer and Eclipse BPEL Designer.

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 following directory. Note that the directory named `default` in these directory paths refers to the domain in which you deployed the suitcase JAR file. If you deploy your BPEL process to a domain named `qa`, that name displays instead of `default`. The concepts of domains are described in later sections of this chapter.

- On JDeveloper BPEL Designer

  `Oracle_Home``\integration\orabpel\domains\default\deploy`

- On Eclipse BPEL Designer

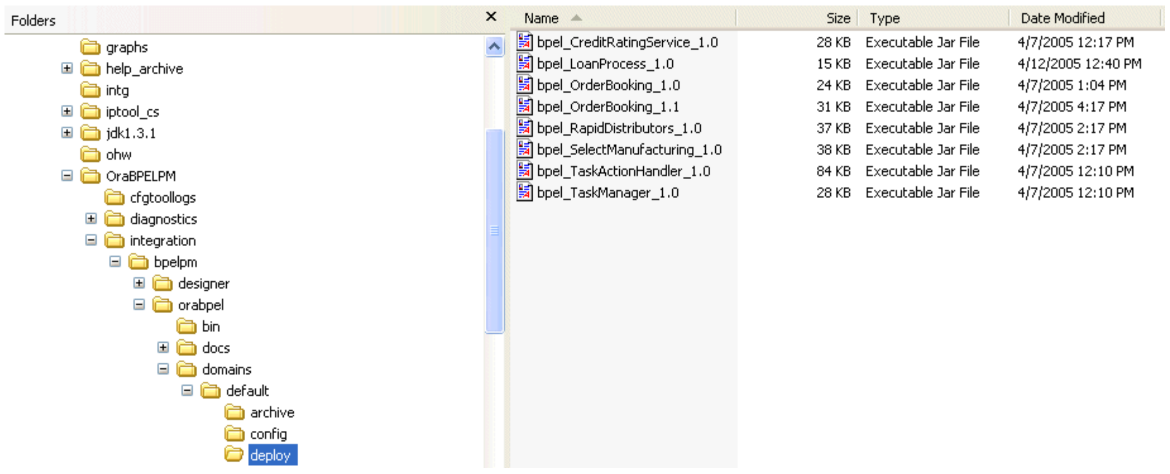
  `Orabpel\domains\default\deploy`

The suitcase JAR file name follows the convention of `bpel_`*`projectname_versionnumber`*`.jar`. For example:

```
bpel_LoanProcess_1.0.jar
```

Figure 19–1 shows an example of deployed suitcase JAR files in JDeveloper BPEL Designer. Note that different version labels of **OrderBooking** are deployed simultaneously.

Figure 19–1 Deployed BPEL Suitcase JAR Files



Name	Size	Type	Date Modified
bpel_CreditRatingService_1.0	28 KB	Executable Jar File	4/7/2005 12:17 PM
bpel_LoanProcess_1.0	15 KB	Executable Jar File	4/12/2005 12:40 PM
bpel_OrderBooking_1.0	24 KB	Executable Jar File	4/7/2005 1:04 PM
bpel_OrderBooking_1.1	31 KB	Executable Jar File	4/7/2005 4:17 PM
bpel_RapidDistributors_1.0	37 KB	Executable Jar File	4/7/2005 2:17 PM
bpel_SelectManufacturing_1.0	38 KB	Executable Jar File	4/7/2005 2:17 PM
bpel_TaskActionHandler_1.0	84 KB	Executable Jar File	4/7/2005 12:10 PM
bpel_TaskManager_1.0	28 KB	Executable Jar File	4/7/2005 12:10 PM

See Also:

- "Creating and Managing a BPEL Domain" on page 19-8
- "Deploying a BPEL Suitcase to a Specific Domain" on page 19-10

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. A BPEL domain is identified by an ID and protected by a password. When Oracle BPEL Process Manager is installed, an initial domain named **default** is created. The initial password of the default domain is set to **bpel**.

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:

- Changing the Default Domain Password
- Changing Oracle BPEL Admin Console Password
- Creating a BPEL Domain
- Changing Oracle BPEL Server Mode
- Deploying a BPEL Suitcase to a Specific Domain
- Location of BPEL JAR Suitcase Files in a Specific Domain
- Undeploying a BPEL Process from a Specific Domain

Changing the Default Domain Password

A domain named **default** is automatically installed with Oracle BPEL Process Manager. The initial password for this domain is **bpel**. You can change this password.

1. Log in to Oracle BPEL Console using Internet Explorer 6.0 by selecting **Start** > **All Programs** > **Oracle - *Oracle_Home*** > **Oracle BPEL Process Manager 10.1.2** > **BPEL Console**.
2. Click **Manage BPEL Domain** at the top of the page.
3. Click **Password**.
4. Enter a new domain password and click **Apply**.

Changing Oracle BPEL Admin Console Password

You create BPEL domains and configure Oracle BPEL Server properties from Oracle BPEL Admin Console. The initial password for Oracle BPEL Admin Console is listed in the *Oracle Application Server Integration Installation Guide*. You must change this password immediately after installation.

Follow these instructions to change Oracle BPEL Admin Console password.

1. Use Internet Explorer 6.0 to access Oracle BPEL Admin Console:

   ```
   http://locahost:9700/BPELAdmin
   ```

You can also access Oracle BPEL Admin Console by clicking **Go to BPEL Admin** at the bottom of the login page for Oracle BPEL Console.

2. Enter the default password.
3. Click the **Password** link in the upper left corner of the window.
4. Enter and re-enter the new administrator password.
5. Enter the existing administrator password.
6. Click **Apply**.

   A message appears indicating that the password has been changed.

Creating a BPEL Domain

You can create additional domains by performing the following procedures.

1. Use Internet Explorer 6.0 to access Oracle BPEL Admin Console:

   ```
   http://locahost:9700/BPELAdmin
   ```

   You can also access Oracle BPEL Admin Console by clicking **Go to BPEL Admin** at the bottom of the login page for Oracle BPEL Console.
2. Enter the 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 and a password. When you log in to Oracle BPEL Console, you can select this domain.
6. Stop and restart Oracle BPEL Server before deploying a BPEL process.
7. Return to the **Applications Navigator.**
8. Right-click a process.
9. Select **Deploy** > ***connection_name*** (for example, **LocalBPELServer**) > **Refresh**.
10. Right-click the process again.
11. Select **Deploy** > ***connection_name*** (for example, **LocalBPELServer**).

    The new domain is now available for selection.

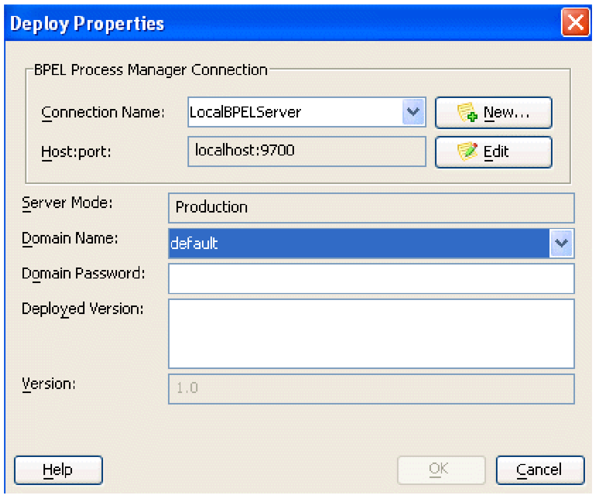
Changing Oracle BPEL Server Mode

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 JDeveloper BPEL Designer.

1. Right-click the BPEL process in the **Applications Navigator**.
2. Select **Deploy** > **Invoke Deployment Tool**.

   The **Server Mode** field of the **Deploy Properties** window displays the mode.



Connection Name: LocalBPELServer
Host:port: localhost:9700
Server Mode: Production
Domain Name: default
Domain Password:
Deployed Version:
Version: 1.0

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 see the current mode of your server in JDeveloper BPEL Designer.

1. Use Internet Explorer 6.0 to access Oracle BPEL Admin Console:

   ```
   http://localhost:9700/BPELAdmin
   ```

   You can also access Oracle BPEL Admin Console by clicking **Go to BPEL Admin** at the bottom of the login page for Oracle BPEL Console.

2. Enter the password.
3. Change the **productionServer** property value to **false**.
4. Click **Apply**.
5. Return to the **Deploy Properties** 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 19-2, there are others methods for deploying a BPEL suitcase into a domain:

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 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 Console to remotely upload and deploy a BPEL JAR. 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 "Location of BPEL JAR Suitcase Files in a Specific Domain" on page 19-11 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 19-23 for additional details about Ant and `bpelc`

Location of BPEL JAR Suitcase Files in a Specific Domain

For a domain named `qa` and a default Oracle BPEL Process Manager installation into the `C:\` directory on Windows, BPEL JAR files for the `qa` domain reside in the following directories:

- On JDeveloper BPEL Designer

  `Oracle_Home``\integration\orabpel\domains\qa\deploy`

- On Eclipse BPEL Designer

  `Orabpel\domains\qa\deploy`

If you use the `bpelc` task with a deploy property value of `qa`, the BPEL suitcase JAR file is automatically copied to this directory. If you want to manually or remotely deploy a BPEL JAR suitcase file to a BPEL domain, you can also copy the JAR file into that directory. The file is automatically detected and loaded into that domain by the Oracle BPEL Process Manager.

See Also: "BPEL Suitcase JAR File" on page 19-6

Undeploying a BPEL Process from a Specific Domain

Oracle BPEL Console 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 Console can also be performed programmatically.

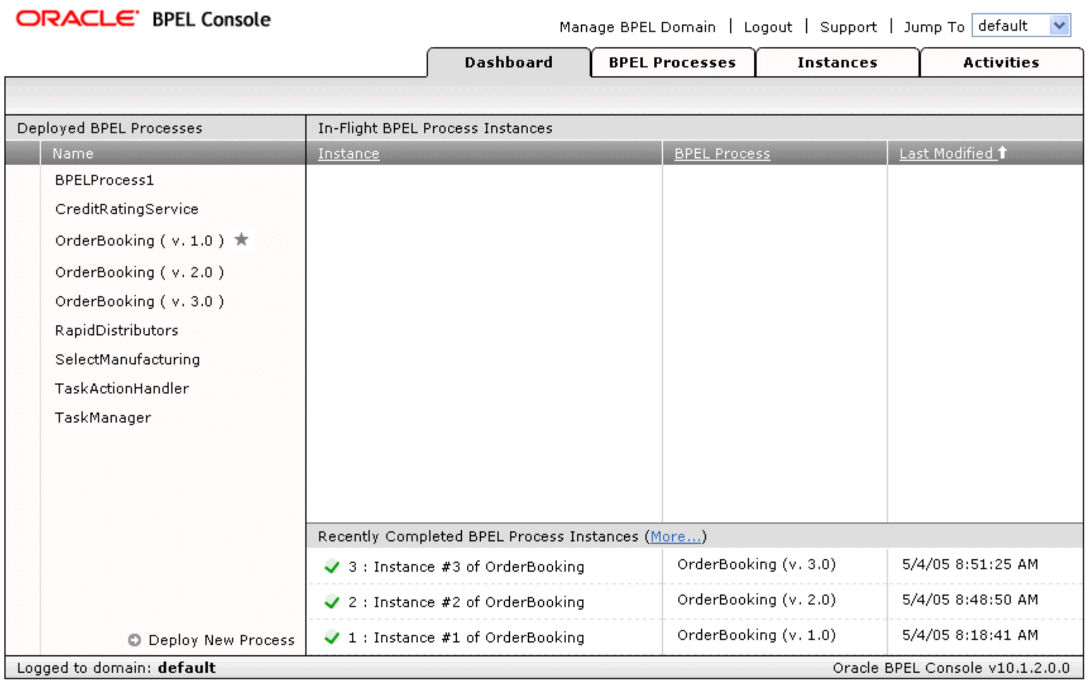
Viewing BPEL Processes in Oracle BPEL Console

If compilation and deployment are successful, you can run and manage the BPEL process from Oracle BPEL Console. This section provides an overview of the main pages of Oracle BPEL Console.

1. Log into Oracle BPEL Console using Internet Explorer 6.0 by selecting **Start** > **All Programs** > **Oracle -** ***Oracle_Home*** > **Oracle BPEL Process Manager 10.1.2** > **BPEL Console**.
2. Select the domain and enter the password. If this is the **default** domain, the password is initially set to **bpel**.
3. See the following sections for an overview of Oracle BPEL Console:
   - Dashboard Tab: Viewing Processes
   - BPEL Processes Tab: Managing the Process Life Cycle
   - Instances Tab: Viewing Process Instances
   - Activities Tab: Viewing Process Activities

Dashboard Tab: Viewing Processes

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 Console to view any currently running BPEL processes before compiling and deploying additional processes. The asterisk indicates that **OrderBooking** version **1.0** is the default process. Default processes are described later in this chapter.



Deployed BPEL Processes

Name
BPELProcess1
CreditRatingService
OrderBooking (v. 1.0) ★
OrderBooking (v. 2.0)
OrderBooking (v. 3.0)
RapidDistributors
SelectManufacturing
TaskActionHandler
TaskManager

Deploy New Process

In-Flight BPEL Process Instances

Instance	BPEL Process	Last Modified

Recently Completed BPEL Process Instances (More...)

✓ 3 : Instance #3 of OrderBooking	OrderBooking (v. 3.0)	5/4/05 8:51:25 AM
✓ 2 : Instance #2 of OrderBooking	OrderBooking (v. 2.0)	5/4/05 8:48:50 AM
✓ 1 : Instance #1 of OrderBooking	OrderBooking (v. 1.0)	5/4/05 8:18:41 AM

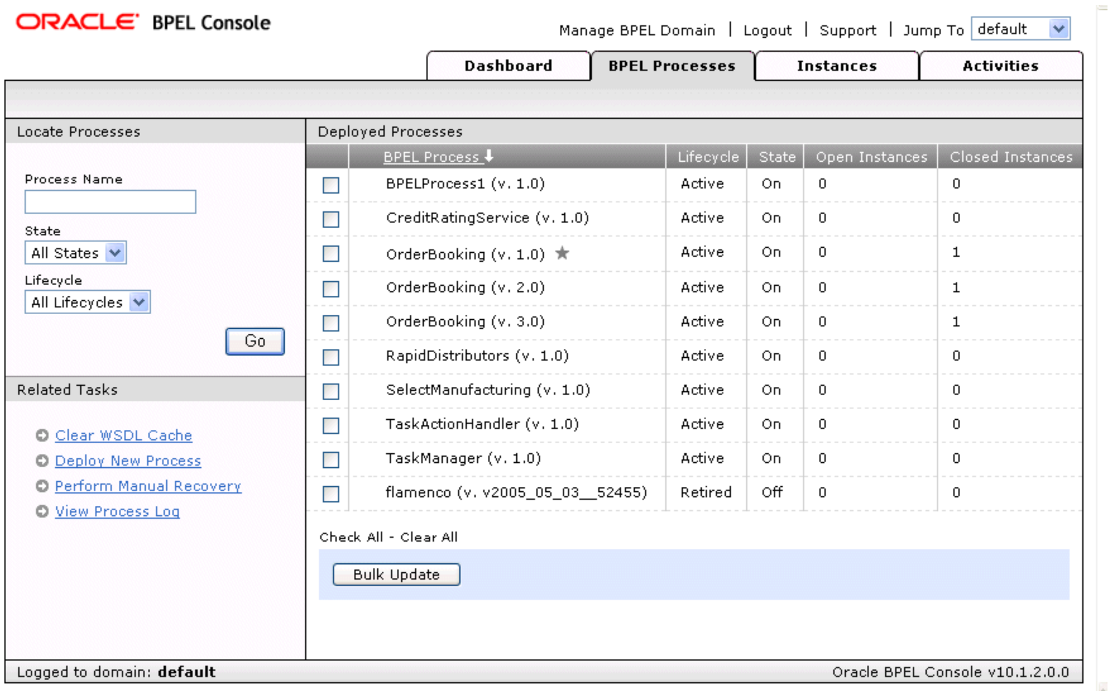
Logged to domain: **default**

Oracle BPEL Console v10.1.2.0.0

Note that each Oracle BPEL Console page includes links at the top for managing BPEL domains and accessing the BPEL site on the Oracle Technology Network, and a drop-down list for switching to another domain.

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 **1.0**) is the default process).



	BPEL Process	Lifecycle	State	Open Instances	Closed Instances
☐	BPELProcess1 (v. 1.0)	Active	On	0	0
☐	CreditRatingService (v. 1.0)	Active	On	0	0
☐	OrderBooking (v. 1.0) ★	Active	On	0	1
☐	OrderBooking (v. 2.0)	Active	On	0	1
☐	OrderBooking (v. 3.0)	Active	On	0	1
☐	RapidDistributors (v. 1.0)	Active	On	0	0
☐	SelectManufacturing (v. 1.0)	Active	On	0	0
☐	TaskActionHandler (v. 1.0)	Active	On	0	0
☐	TaskManager (v. 1.0)	Active	On	0	0
☐	flamenco (v. v2005_05_03__52455)	Retired	Off	0	0

Logged to domain: **default**

Oracle BPEL Console v10.1.2.0.0

For each BPEL process, Oracle BPEL Console shows the following status indicators:

- 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

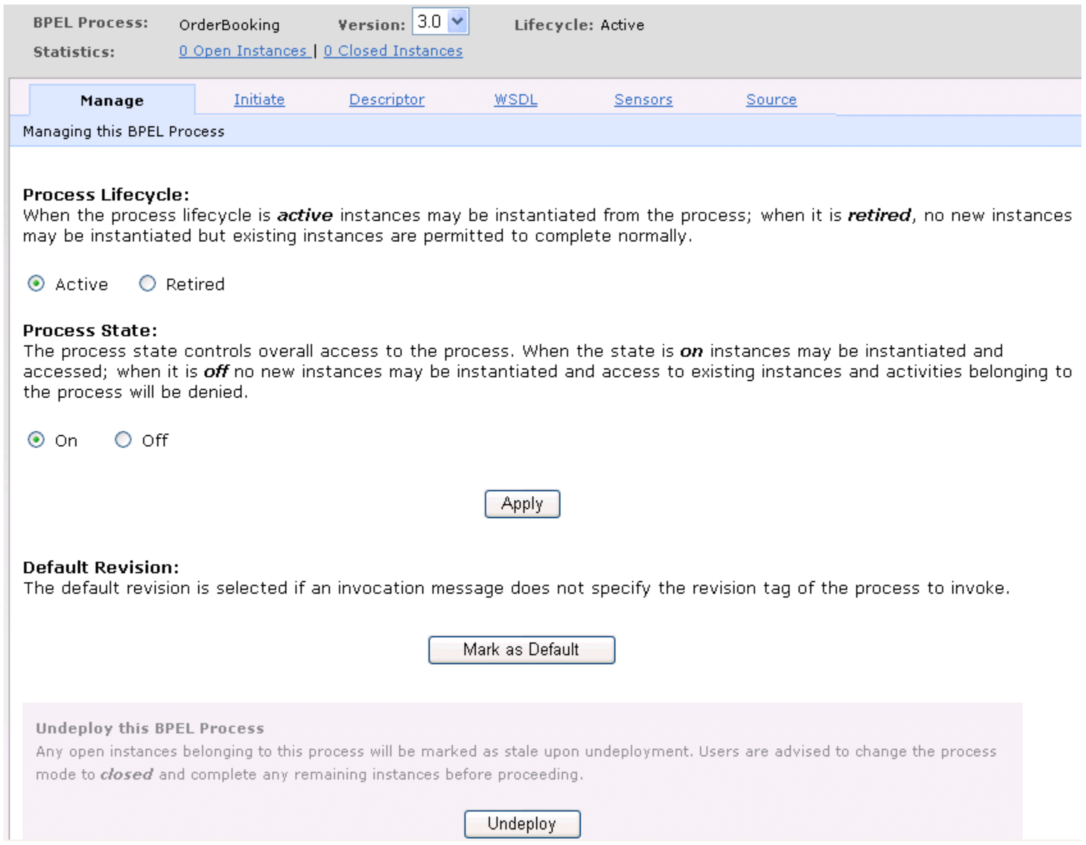
  The number of open instances. An open instance is an instance that is currently being processed.

- Completed Instances

  The number of completed instances. A completed instance is an instance that has completed processing, either successfully or due to an error.

2. Click a specific process in the **BPEL Process** list (for this example, **OrderBooking (v. 1.0)**)

   This window enables you to manage the life cycle and state of the BPEL process.


BPEL Process: OrderBooking Version: 3.0 Lifecycle: Active
Statistics: 0 Open Instances | 0 Closed Instances

Managing this BPEL Process

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*** instances may be instantiated and accessed; when it is ***off*** no new instances may be instantiated and access to existing instances and activities belonging to the process will be denied.

[✓ On] Off

Default Revision:
The default revision is selected if an invocation message does not specify the revision tag of the process to invoke.

Undeploy this BPEL Process
Any open instances belonging to this process will be marked as stale upon undeployment. Users are advised to change the process mode to ***closed*** and complete any remaining instances before proceeding.

3. 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
4. Ensure that you understand the following process life cycle and state concepts:

Process	Description
Process Life Cycles	
▪ Active	All processes when deployed are automatically active (that is, existing versions are not automatically retired). You must explicitly retire processes.
▪ Retired	A process that is no longer used. When a process is retired, all currently executing instances complete normally. You can view previously completed instances.
Process State	
▪ On	Process instances can be instantiated and accessed.
▪ Off	Process instances cannot be instantiated and accessed. Access to existing instances and activities of the process is not allowed.

Process	Description
Default Revision	A designated process and revision that is instantiated when a new request comes in. The default process is identified by an asterisk next to its name in Oracle BPEL Console. When deployed, a process is marked as default only if it is the first version of a deployed process. There can be only one default process. Subsequent deployments of an already-deployed process do not become defaults by virtue of deployment. 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.
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.

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 2 on page 19-13. 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.

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.

Example: Life Cycle of Processes

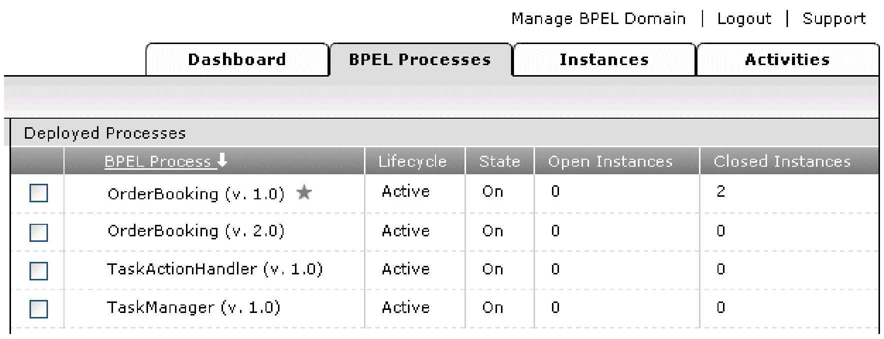
This section provides a brief example of the various life cycle states of two process versions. In the first stage, two versions of the same processes are created, as shown in shown in Table 19–1. **OrderBooking** version 1.0 receives two messages, which results in the creation of two instances.

Table 19–1 Stage 1: Two Process Versions Created

Stage	Action	Life Cycle	State	Default Process	On Arrival of New Message Request
1	Deploy **OrderBooking** version 1.0	Active=1.0	On=1.0	Version 1.0 (automatically set)	Create an instance for **OrderBooking** version 1.0
1	Deploy **OrderBooking** version 2.0	Active=2.0	On=2.0	Version 1.0	Create another instance for **OrderBooking** version 1.0

The life cycle and state of the two **OrderBooking** versions displays in the **BPEL Processes** tab shown in Figure 19–2. Because **OrderBooking** version 1.0 was the first deployed version of this process, it automatically became the default process (indicated by the asterisk). The two messages that resulted in the creation of two **OrderBooking** version 1.0 instances have both completed.

Figure 19–2 Stage 1 BPEL Processes

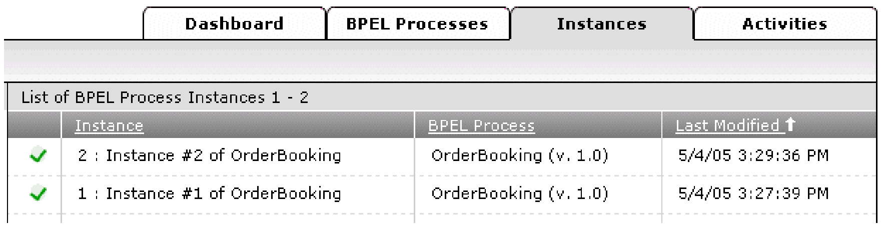


Deployed Processes

	BPEL Process	Lifecycle	State	Open Instances	Closed Instances
☐	OrderBooking (v. 1.0) ★	Active	On	0	2
☐	OrderBooking (v. 2.0)	Active	On	0	0
☐	TaskActionHandler (v. 1.0)	Active	On	0	0
☐	TaskManager (v. 1.0)	Active	On	0	0

The two completed instances of **OrderBooking** version 1.0 display in the **Instances** tab shown in Figure 19–3.

Figure 19–3 Stage 1 BPEL Instances



List of BPEL Process Instances 1 - 2

	Instance	BPEL Process	Last Modified
✓	2 : Instance #2 of OrderBooking	OrderBooking (v. 1.0)	5/4/05 3:29:36 PM
✓	1 : Instance #1 of OrderBooking	OrderBooking (v. 1.0)	5/4/05 3:27:39 PM

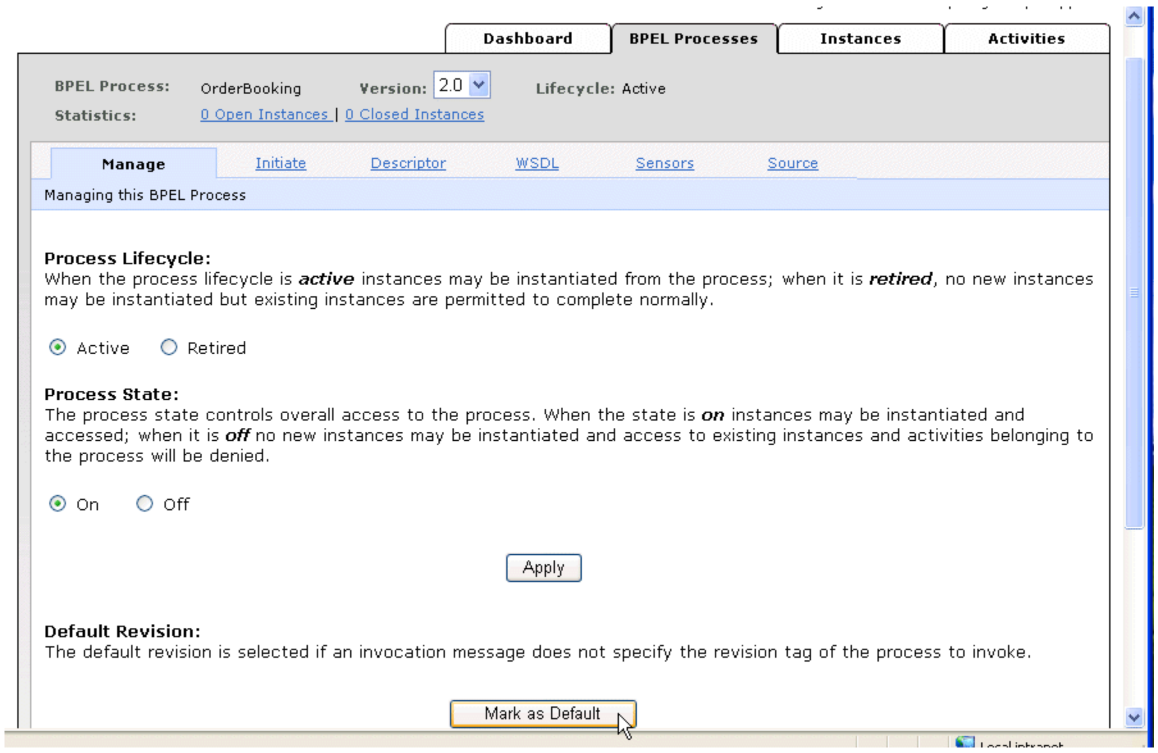
In stage 2, you change **OrderBooking** version 2.0 to be the default and retire **OrderBooking** version 1.0, as shown in Table 19–2.

Table 19–2 Stage 2 Change the Default Process and Retire a Process

Stage	Action	Life Cycle	State	Default Process	On Arrival of New Message Request
1	Deploy **OrderBooking** version 1.0	Active=1.0	On=1.0	Version 1.0 (automatically set)	Create an instance for **OrderBooking** version 1.0
1	Deploy **OrderBooking** version 2.0	Active=2.0	On=2.0	Version 1.0	Create another instance for **OrderBooking** version 1.0
2	Change default process to **OrderBooking** version 2.0	Active=1.0	On=1.0	Version 2.0	Create an instance for **OrderBooking** version 2.0
2	Retire **OrderBooking** version 1.0	Retired=2.0	On=2.0	Version 2.0	No action

Figure 19–4 shows **Mark as Default** being selected for **OrderBooking** version 2.0.

Figure 19–4 Stage 2 Default Revision



BPEL Process: OrderBooking Version: 2.0 Lifecycle: Active
Statistics: 0 Open Instances | 0 Closed Instances

Managing this BPEL Process

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

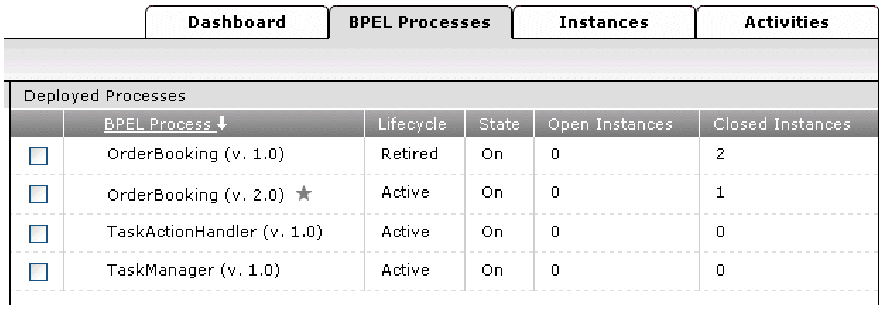
Process State:
The process state controls overall access to the process. When the state is ***on*** instances may be instantiated and accessed; when it is ***off*** no new instances may be instantiated and access to existing instances and activities belonging to the process will be denied.

[✓ On] Off

Default Revision:
The default revision is selected if an invocation message does not specify the revision tag of the process to invoke.

The modified life cycle and state of the two **OrderBooking** versions displays in the **BPEL Processes** tab shown in Figure 19–5. Because **OrderBooking** version 2.0 was explicitly selected as the default process, it now displays the asterisk. The message that resulted in the creation of an **OrderBooking** version 1.0 instance has completed. **OrderBooking** version 1.0 displays as **Retired**.

Figure 19–5 Stage 2 BPEL Processes

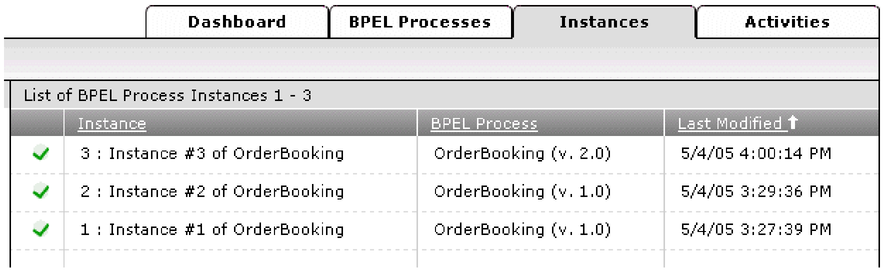


Deployed Processes

	BPEL Process	Lifecycle	State	Open Instances	Closed Instances
☐	OrderBooking (v. 1.0)	Retired	On	0	2
☐	OrderBooking (v. 2.0) ★	Active	On	0	1
☐	TaskActionHandler (v. 1.0)	Active	On	0	0
☐	TaskManager (v. 1.0)	Active	On	0	0

The completed instance of **OrderBooking** version 2.0 displays in the **Instances** tab shown in Figure 19–6.

Figure 19–6 Stage 2 BPEL Instances



List of BPEL Process Instances 1 - 3

	Instance	BPEL Process	Last Modified
✓	3 : Instance #3 of OrderBooking	OrderBooking (v. 2.0)	5/4/05 4:00:14 PM
✓	2 : Instance #2 of OrderBooking	OrderBooking (v. 1.0)	5/4/05 3:29:36 PM
✓	1 : Instance #1 of OrderBooking	OrderBooking (v. 1.0)	5/4/05 3:27:39 PM

If you click the **Dashboard** tab, the retired **OrderBooking** version 1.0 no longer appears. This means you can no longer create an instance for this version.

In stage 3, you make **OrderBooking** version 1.0 inactive and then undeploy it, as shown in Table 19–3.

Table 19–3 Stage 3 Deactivate and Undeploy a Process

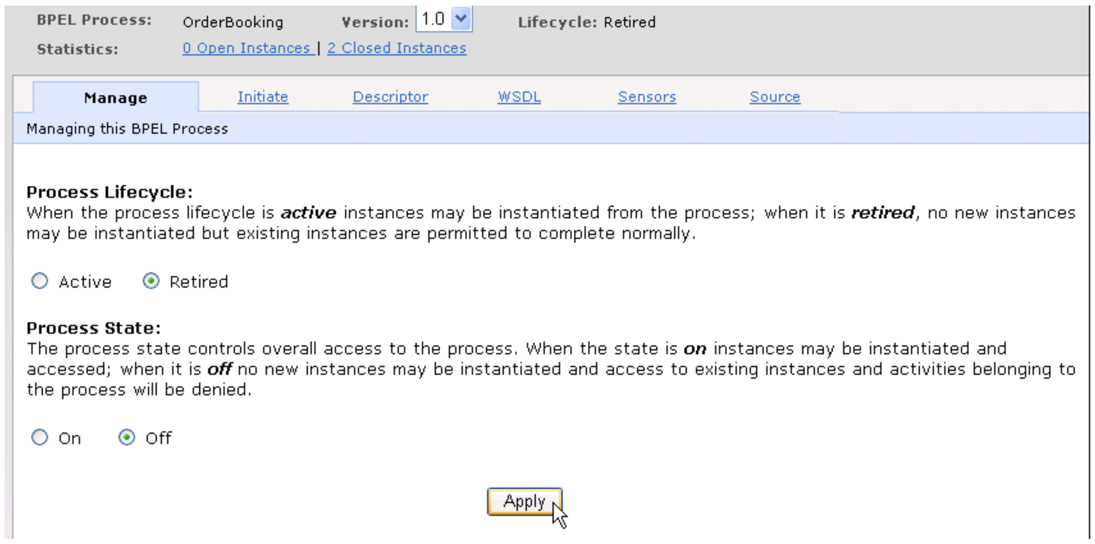
Stage	Action	Life Cycle	State	Default Process	On Arrival of New Message Request
1	Deploy **OrderBooking** version 1.0	Active=1.0	On=1.0	Version 1.0 (automatically set)	Create an instance for **OrderBooking** version 1.0
1	Deploy **OrderBooking** version 2.0	Active=2.0	On=2.0	Version 1.0	Create another instance for **OrderBooking** version 1.0
2	Change default process to **OrderBooking** version 2.0	Active=1.0	On=1.0	Version 2.0	Create an instance for **OrderBooking** version 2.0
2	Retire **OrderBooking** version 1.0	Retired=2.0	On=2.0	Version 2.0	No action
3	Make **OrderBooking** version 1.0 inactive	Retired=2.0	Off=2.0	Version 2.0	No action

Table 19–3 (Cont.) Stage 3 Deactivate and Undeploy a Process

Stage	Action	Life Cycle	State	Default Process	On Arrival of New Message Request
3	Undeploy **OrderBooking** version 1.0	Retired=2.0	Off=2.0	Version 2.0	No action

The state of **OrderBooking** version 1.0 is changed to **Off**.

Figure 19–7 Stage 3 Process State



BPEL Process: OrderBooking Version: 1.0 Lifecycle: Retired
Statistics: 0 Open Instances | 2 Closed Instances

Manage | Initiate | Descriptor | WSDL | Sensors | Source

Managing this BPEL Process

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]

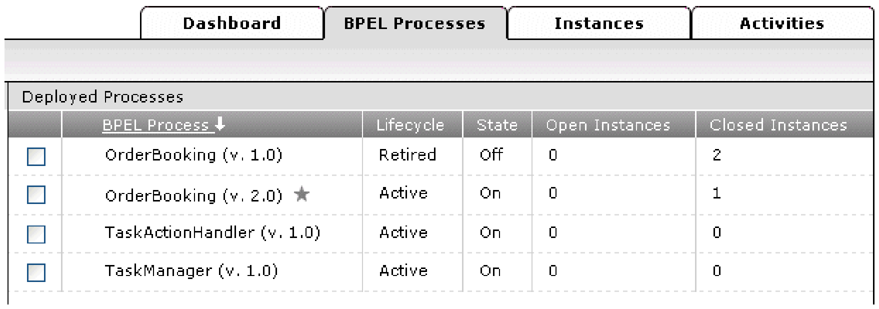
Process State:
The process state controls overall access to the process. When the state is ***on*** instances may be instantiated and accessed; when it is ***off*** no new instances may be instantiated and access to existing instances and activities belonging to the process will be denied.

On [✓ Off]

Apply

The state of **OrderBooking** version 1.0 displays as **Off** in the **BPEL Processes** tab shown in Figure 19–8. 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.

Figure 19–8 Stage 3 BPEL Processes



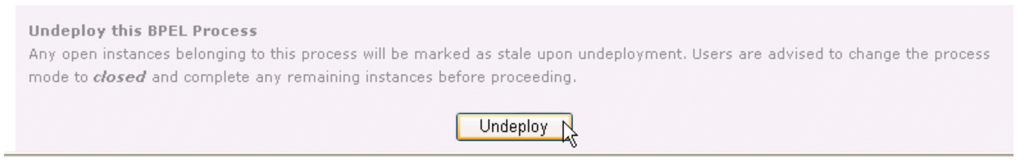
Dashboard | BPEL Processes | Instances | Activities

Deployed Processes

	BPEL Process	Lifecycle	State	Open Instances	Closed Instances
☐	OrderBooking (v. 1.0)	Retired	Off	0	2
☐	OrderBooking (v. 2.0) ★	Active	On	0	1
☐	TaskActionHandler (v. 1.0)	Active	On	0	0
☐	TaskManager (v. 1.0)	Active	On	0	0

OrderBooking version 1.0 is then undeployed, as shown in Figure 19–9.

Figure 19–9 Stage 3 Undeploy

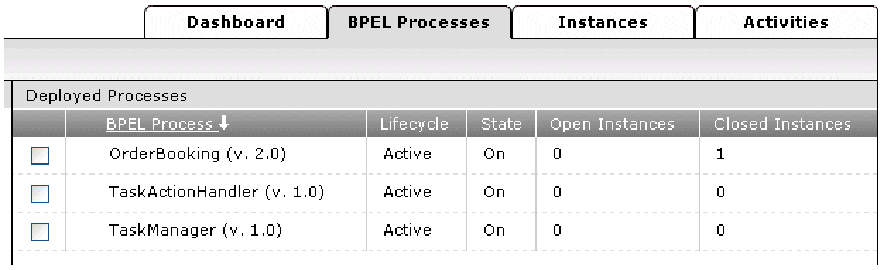


Undeploy this BPEL Process

Any open instances belonging to this process will be marked as stale upon undeployment. Users are advised to change the process mode to ***closed*** and complete any remaining instances before proceeding.

The **BPEL Processes** tab in Figure 19–10 no longer displays **OrderBooking** version 1.0. Note also that the asterisk no longer displays for **OrderBooking** 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. This indicates that this is the default process. If you deploy an **OrderBooking** version 3.0, the asterisk again appears next to **OrderBooking** version 2.0.

Figure 19–10 Stage 3 BPEL Processes

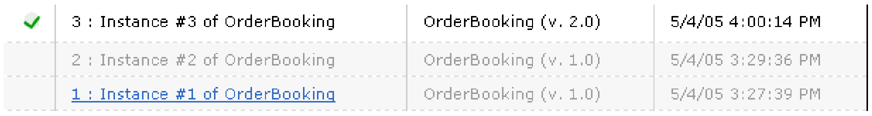


Deployed Processes

	BPEL Process	Lifecycle	State	Open Instances	Closed Instances
☐	OrderBooking (v. 2.0)	Active	On	0	1
☐	TaskActionHandler (v. 1.0)	Active	On	0	0
☐	TaskManager (v. 1.0)	Active	On	0	0

The two completed instances of the undeployed **OrderBooking** version 1.0 display as disabled in the **Instances** tab shown in Figure 19–11.

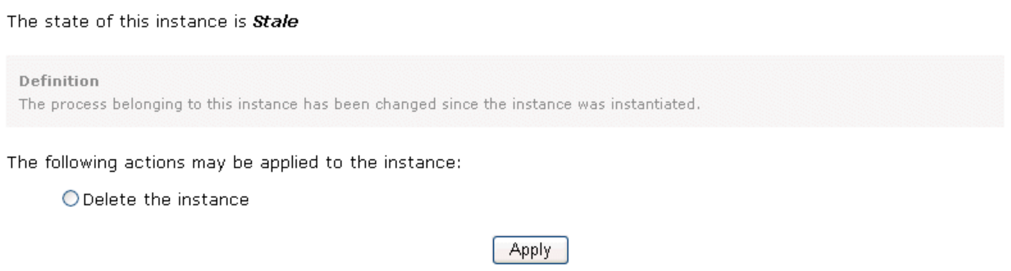
Figure 19–11 Stage 3 Instance of Undeployed Process



✓	3 : Instance #3 of OrderBooking	OrderBooking (v. 2.0)	5/4/05 4:00:14 PM
	2 : Instance #2 of OrderBooking	OrderBooking (v. 1.0)	5/4/05 3:29:36 PM
	1 : Instance #1 of OrderBooking	OrderBooking (v. 1.0)	5/4/05 3:27:39 PM

Clicking one of the completed instances displays the status as **Stale** in Figure 19–12.

Figure 19–12 Stage 3 Status Message of Undeployed Process



The state of this instance is ***Stale***

Definition

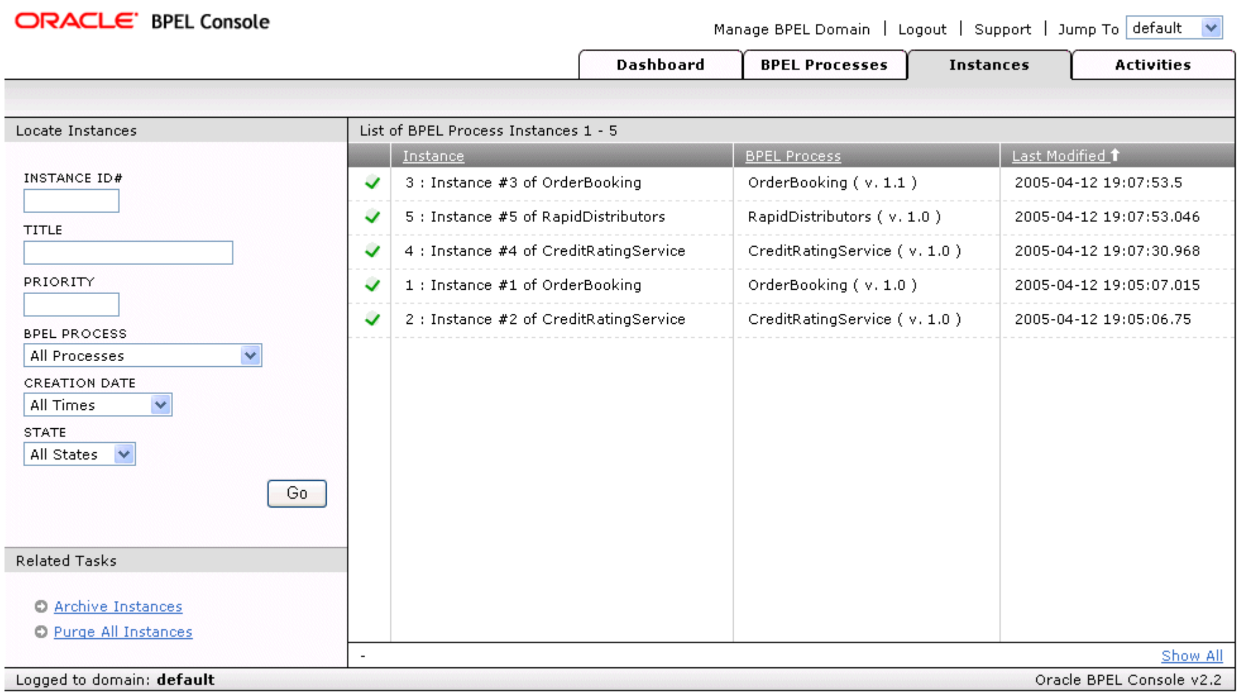
The process belonging to this instance has been changed since the instance was instantiated.

The following actions may be applied to the instance:

○ Delete the instance

Instances Tab: Viewing Process Instances

1. Click the **Instances** tab to view BPEL process instances.



List of BPEL Process Instances 1 - 5

	Instance	BPEL Process	Last Modified
✓	3 : Instance #3 of OrderBooking	OrderBooking (v. 1.1)	2005-04-12 19:07:53.5
✓	5 : Instance #5 of RapidDistributors	RapidDistributors (v. 1.0)	2005-04-12 19:07:53.046
✓	4 : Instance #4 of CreditRatingService	CreditRatingService (v. 1.0)	2005-04-12 19:07:30.968
✓	1 : Instance #1 of OrderBooking	OrderBooking (v. 1.0)	2005-04-12 19:05:07.015
✓	2 : Instance #2 of CreditRatingService	CreditRatingService (v. 1.0)	2005-04-12 19:05:06.75

2. Click an instance in the **Instance** column (for example, **Instance #3 of OrderBooking**). From this page, you can perform the following tasks:

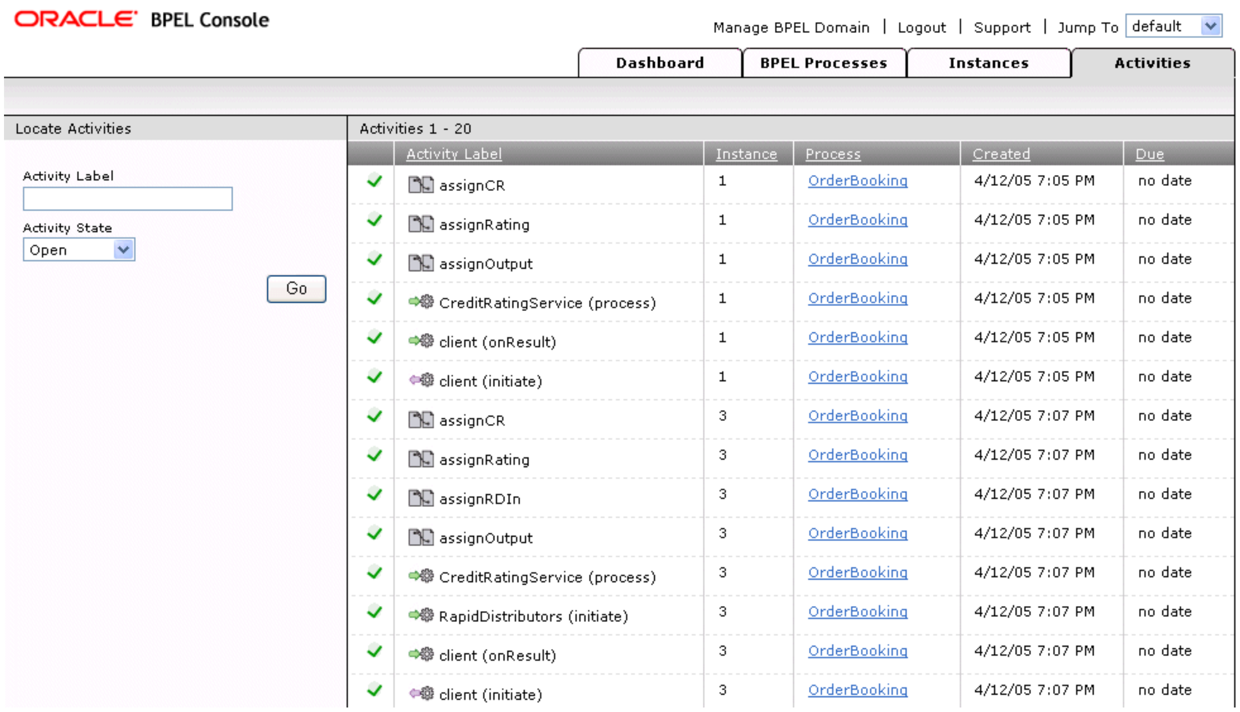
   - View the state of the instance (for example, ***COMPLETED*** or ***ACTIVE***)
   - Click **Flow** to view a visual representation of the history of the activities in this instance.
   - Click **Audit** to view an audit trail of this instance.
   - Click **Debug** to view the code of the BPEL file.
   - 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.

   See Also: The following documentation for additional details about sensors:

   - Chapter 18, "Sensors"
   - *Oracle BPEL Process Manager Order Booking Tutorial*

Activities Tab: Viewing Process Activities

1. Click the **Activities** tab to view the status of activities in the deployed BPEL process instance.



Activities 1 - 20

	Activity Label	Instance	Process	Created	Due
✓	assignCR	1	OrderBooking	4/12/05 7:05 PM	no date
✓	assignRating	1	OrderBooking	4/12/05 7:05 PM	no date
✓	assignOutput	1	OrderBooking	4/12/05 7:05 PM	no date
✓	CreditRatingService (process)	1	OrderBooking	4/12/05 7:05 PM	no date
✓	client (onResult)	1	OrderBooking	4/12/05 7:05 PM	no date
✓	client (initiate)	1	OrderBooking	4/12/05 7:05 PM	no date
✓	assignCR	3	OrderBooking	4/12/05 7:07 PM	no date
✓	assignRating	3	OrderBooking	4/12/05 7:07 PM	no date
✓	assignRDIn	3	OrderBooking	4/12/05 7:07 PM	no date
✓	assignOutput	3	OrderBooking	4/12/05 7:07 PM	no date
✓	CreditRatingService (process)	3	OrderBooking	4/12/05 7:07 PM	no date
✓	RapidDistributors (initiate)	3	OrderBooking	4/12/05 7:07 PM	no date
✓	client (onResult)	3	OrderBooking	4/12/05 7:07 PM	no date
✓	client (initiate)	3	OrderBooking	4/12/05 7:07 PM	no date

See Also: The following documentation for tutorials in which you run processes from Oracle BPEL Console and view their results from the **Audit** and **Flow** links:

- "Testing the BPEL Process" on page 3-17
- *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:

- Apache Ant
- bpelc
- schemac

Apache Ant

Apache 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. For example, when you compile and deploy a BPEL process, you see `bpelc` being invoked as a task inside the Apache Ant `build.xml` file in the **Log Window** of Eclipse BPEL Designer.

Table 19–4 shows the supported `bpelc` options:

Table 19–4 Parameters

Attribute	Description	Required
`home`	The `orabpel` home directory (or whatever you named your *`Oracle_Home`* directory), which is typically available as Ant property `$home`	No
`input`	The deployment descriptor location path, By default, it looks for `bpel.xml` under the current directory.	No
`rev`	The revision (version) tag for the deployed process.	No
`deploy`	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.	No
`keepGenerated`	Includes the BPEL process Java classes in the generated BPEL archive. The value defaults to `false`.	No
`verbose`	Generates additional debugging messages about compiler actions. The value defaults to `false`.	No
`force`	Always compiles the process; the compiler does not check the time stamp of `.bpel`, `.wsdl` and `.xml` files. The value defaults to `false`.	No
`classpath`	Specifies where to find user class files. This attribute is similar to a `PATH` structure and can also be set through a nested class path element.	No
`lib`	Oracle BPEL Process Manager system `lib` directory.	No
`help`	Displays the help message. This value defaults to `false`.	No

Table 19–4 (Cont.) Parameters

Attribute	Description	Required
`out`	Specifies the location in which to deploy the BPEL archive. This option is used when the `deploy` attribute is not used. For example: `out="c:\myproject\bpel\deploy"`	No

Examples

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:

```
<bpelc home="${home}" rev="${rev}" deploy="default"/>
```

To deploy the BPEL archive into the `c:\myproject` directory:

```
<bpelc home="${home}" rev="${rev}" out="C:\myproject"/>
```

Specify a deployment descriptor file name:

```
<bpelc home="${home}" rev="${rev}" deploy="default" input="orderdd.xml"/>
```

Specify a user `classpath` for `bpelc`:

```
<bpelc home="${home}" rev="${rev}" deploy="default"/>
<classpath>
<pathelement location="dist/test.jar"/>
<pathelement path="${java.class.path}"/>
</classpath>
</bpelc>
```

schemac

`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 `complexTypes` schema types and element with an anonymous `complexType`. This is similar to the `jaxb` compiler.

You 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 19–5 describes the supported options:

Table 19–5 Parameters

Attribute	Description	Required
`input`	The XML schema is the name of a file (ending with `.xsd` or `.wsdl`) containing a valid XML schema definition.	Yes

Table 19–5 (Cont.) Parameters

Attribute	Description	Required
`out`	Specify where to place generated facade class files. This value defaults to the current directory.	No
`doc`	Generates Javadoc for the generated classes and redirects it to the specified location.	No
`jar`	Archives the generated classes into the specified JAR file name.	No
`verbose`	Generates more debugging messages about the compiler actions. Defaults to `false`.	No
`noCompile`	Generates only the Java source files and does not compile the generated sources when set to `true`. This value defaults to `false`.	No
`help`	Displays the help message. This value defaults to `false`.	No
`sourceOut`	Specifies the location in which to redirect the generated Java files. For example: `sourceOut=”c:\myproject\bpel\facade\source”`	No
`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.goo` **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`.	No

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="${basedir}/Order.xsd " out="${basedir}/BPEL-INF/classes"
nsMap="mynsmap.txt"/>
```

See Also:

- "XML Facade" on page 10-3
- *Oracle_ Home*`\integration\orabpel\samples\tutorials\702.Bindings` for XML facade samples

Summary

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 changing domain and BPEL Admin Console passwords, creating domains, changing Oracle BPEL Server modes (production or development), managing BPEL suitcase JAR files, and undeploying processes. An overview of Oracle BPEL Console is also provided, including a detailed description of managing different versions of BPEL processes. Finally, a discussion on how to use the Apache Ant, bpelc, and schemac build tools is provided.

Part V

Reference Information

This part provides reference details about troubleshooting issues, activities, deployment descriptor properties, demo user communities, and XPath extension functions.

This part contains the following appendices:

- Appendix A, "Troubleshooting and Workarounds"
- Appendix B, "Workflow and Notification Reference"
- Appendix C, "JDeveloper BPEL Designer Activities"
- Appendix D, "User Task 2.0 Macro"
- Appendix E, "Deployment Descriptor Properties"
- Appendix F, "Demo User Community"
- Appendix G, "XPath Extension Functions"

A Troubleshooting and Workarounds

This appendix describes Oracle BPEL Process Manager troubleshooting methods.

This appendix contains the following topics:

- Troubleshooting Sensors—The Custom Data Publisher
- Troubleshooting Oracle BPEL Worklist Application
- Summary

Troubleshooting Sensors—The Custom Data Publisher

The following sections describe possible issues and solutions.

Poor JMS Performance When Creating or Destroying Connections

Problem

Due to a bug, you can experience poor JMS performance when creating or destroying connections. This can cause a slowdown in the execution of BPEL processes that have JMS data publishers associated with them.

Solution

Use the rollup patch included on the software CD. See the readme file.

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.
- 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 Console 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 JAX-B 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 17-33 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. See "Identity Service" on page 16-75 for more information.

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 admin) 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.

B Workflow and Notification Reference

This appendix describes operations that can be invoked from a BPEL business process.

This appendix contains the following topics:

- Task Manager Service WSDL Operations
- Task Routing Service WSDL Operations
- Notification WSDL Operations
- Identify Service Operations
- Task Action Handler Business Process
- Summary

Task Manager Service WSDL Operations

The task manager service exposes operations that can be invoked from the BPEL business process to orchestrate the workflow. The task manager service also provides state management and persistency for the tasks. The task manager service exposes the following operations. All the operations are exposed on the WSDL port type `TaskManager`.

- `initiateTask`—initiates a task by persisting the task in the database
- `reinitiateTask`—reinitiates a task to continue an existing task
- `updateTask`—updates the task
- `renewTask`—renews the task with a specified duration
- `notifyTaskExpiration`—notifies the task manager service that the task is expired. This operation is invoked from the task action handler business process, which manages the expiration of the task.
- `initiateSubTask`—initiates a subtask in a parallel workflow pattern
- `completeTask`—completes the task with a specified conclusion
- `sendTaskReminder`—notifies the task manager service to send a reminder for the task. This operation is invoked from the task action handler business process, which manages task reminders.
- `releaseTask`—releases a previously acquired task
- `errorTask`—notifies the task manager service that the task is in an errored state and that the task should be marked as errored

The WSDL for `TaskManagerService` is available at

```
http://hostname:port/orabpel/xmllib/workflow/TaskManagerService.wsdl
```

The XSD used by `TaskManagerService` is available at

```
http://hostname:port/orabpel/xmllib/workflow/WorkflowTask.xsd
```

Task Routing Service WSDL Operations

The task routing service exposes operations that can be invoked from the BPEL business process to perform task routing operations. The task manager service exposes the following operations. All the operations are exposed on the WSDL port type `TaskRoutingService`.

- `routeTask`—routes the task to a specified participant
- `routeTaskToNextApprover`—routes the task to the next approver as determined by evaluating the approver function
- `escalateTask`—escalates the task to the manager
- `escalateTaskOnExpiration`—escalates the task when the task is expired. The operation also renews the task by a specified duration.

The WSDL for `TaskRoutingService` is available at

```
http://hostname:port/orabpel/default/TaskActionHandler/TaskRoutingService.wsdl
```

The XSD for the service is available at

```
http://hostname:port/orabpel/xmllib/workflow/TaskRoutingService.xsd
```

Notification WSDL Operations

The notification service exposes operations that can be invoked from the BPEL business process to send notifications through e-mail, voice, or short message service (SMS) channels. The notification service exposes the following operations on the WSDL port type `NotificationService`.

- `sendVoiceNotification`—sends an instance messenger notification
- `sendSMSNotification`—sends an instance messenger notification
- `sendEmailNotification`—sends an e-mail notification
- `sendNotificationToUser`—sends a notification to a user. The notification channel used is determined by the `orclWorkflowNotificationPref` attribute in Oracle Internet Directory (OID) or JAZN-XML. If no preference is specified, then e-mail is used. The input message to this operation contains two parts—the group ID and the generic notification payload. The generic payload contains a common payload that applies to all the channels and a payload for each of the channels. The message that is actually sent is evaluated from the generic payload.
- `sendNotificationToGroup`—sends a notification to a group using the e-mail channel. If the group does not have an e-mail address associated with it, then the notification is sent to each member of the group using their preferred notification channel. The input message to this operation contains two parts—the group ID and the generic notification payload. The generic payload contains a common payload that applies to all the channels and a payload for each of the channels. The message that is actually sent is evaluated from the generic payload.

Identify Service Operations

The identity service exposes the following operations that can be invoked from the BPEL business process. All the operations are exposed on the WSDL port type `IdentityService`.

- `getManager`—gets the manager of a user, which can also be obtained using the `ora:getManager()` XPath extension function.
- `getManagementChain`—gets the management chain based on the input message

The WSDL for the `IdentityService` is available at

```
http://hostname:port/orabpel/xmllib/workflow/LocalIdentityService.wsdl
```

The XSD for the service is available at

```
http://hostname:port/orabpel/xmllib/workflow/LocalIdentityService.xsd
```

Task Action Handler Business Process

The task action handler business process receives updates from Oracle BPEL Worklist Application through the task manager service or the task routing service. The main task of the process is to keep the task synchronized in the BPEL process with its database state. Based on the state of the task, the process determines if the parent BPEL process should be called back with the task. An instance of the task action handler process is created for each participant for a given task. Participants who participate in the task when the current participant performs operations like reassign, escalate, and so on reuse the same process.

In addition, the task action handler process manages the following timer events:

- Task expiration
- Reminder durations for the task
- Task release durations when the task is acquired

Summary

This appendix describes workflow and notification service WSDL operations.

C
JDeveloper BPEL Designer Activities

This appendix describes the activities available for use when designing a BPEL process in JDeveloper BPEL Designer.

This appendix contains the following topics:

- Validating when Loading a Process Diagram
- Activities Overview
- Summary

Validating when Loading a Process Diagram

As you create and open activities such as scope, assign, and others for the first time in JDeveloper BPEL Designer, the message `Invalid Settings` appears at the top of the activity window. This is because you have not yet entered details. If the **Validate process when loading diagram** option is selected, the message always appears. You can ignore this message. As you enter and correctly apply your details, the message disappears. If you want to turn this option off for the current project, perform the following steps:

1. Right-click in the JDeveloper BPEL Designer **Diagram View** and select **Diagram Properties**.
2. Deselect the **Validate process when loading diagram option** on the Diagram Properties window.
3. Click **OK**.
4. Select **Save All** from the **File** main menu.

Activities Overview

JDeveloper BPEL Designer includes a series of 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:

- Assign Activity
- Catch Activity
- Compensate Activity
- Empty Activity
- Flow Activity
- FlowN Activity
- Invoke Activity
- Java Embedding Activity
- Notification Activity
- PartnerLink Activity
- Pick Activity
- Receive Activity
- Reply Activity
- Sequence Activity
- Scope Activity
- Switch Activity
- Terminate Activity
- Throw Activity
- Transform Activity
- User Task

- 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*
> - The contents of the `c:\Oracle_Home\integration\orabpel\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 **Correlation Sets** 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, and Java Message Service (JMS) adapters.

> **See Also:**
>
> - The Online help for these tabs for additional details about their use
> - Chapter 18, "Sensors"
> - *Oracle Adapters for Files, FTP, Databases, and Enterprise Messaging User's Guide*

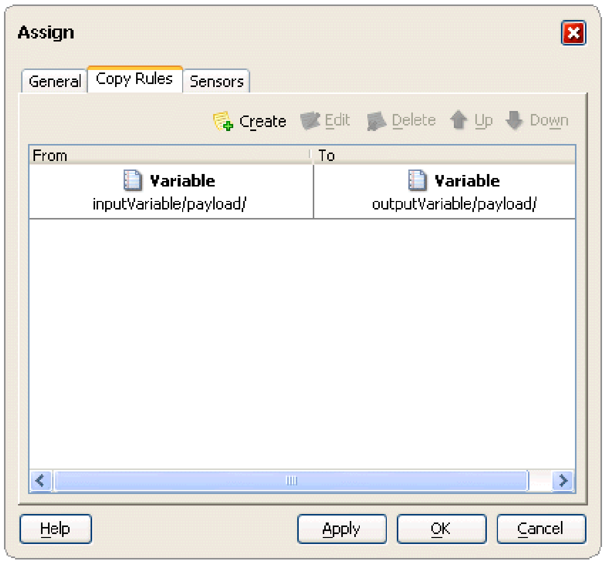
Assign Activity

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 Rules** tab and the **Create** icon shown in Figure C–1 to access the Create Copy Rule window. 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).

Figure C–1 Copy Rules Tab of Assign Activity WIndow



From	To
Variable inputVariable/payload/	Variable outputVariable/payload/

See Also: The following documentation for many examples of using the **Assign** activity:

- *Oracle BPEL Process Manager Order Booking Tutorial*
- *Oracle BPEL Process Manager Quick Start Guide*
- `*Oracle_Home*\integration\orabpel\samples\references\Assign`

Catch Activity

This activity enables you to create optional fault handling logic to catch and manage exceptions. Fault handling is associated with a **scope** activity. Fault handling's goal is to undo the incomplete and unsuccessful work of a **scope** activity in which a fault has occurred.

Fault handlers in a **scope** activity enable you to create a set of custom fault-handling activities, which are defined as **catch** activities. Each **catch** activity is defined to intercept a specific type of fault.

Figure C–2 shows the **Add Catch Branch** icon inside a **scope** activity that you click to create a **catch** activity. Figure C–3 shows an example of a **catch** activity on the right side of the **scope** activity. 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.

Figure C–2 Creating an Add Catch Activity

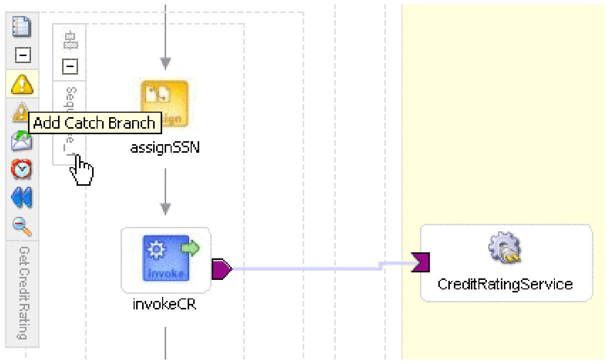
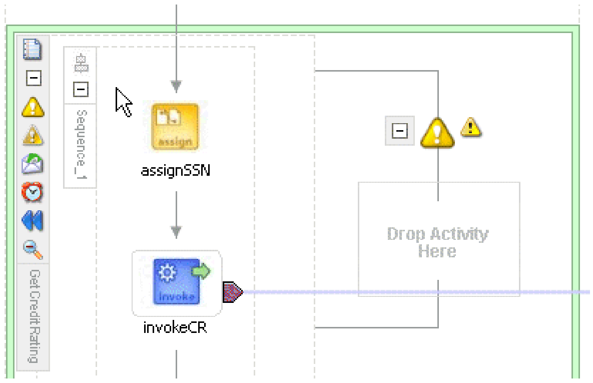


Figure C–3 Catch Activity Icon



See Also: The following documentation for many examples of using the **catch** activity:

- *Oracle BPEL Process Manager Quick Start Guide*
- *Oracle BPEL Process Manager Order Booking Tutorial*
- `Oracle_Home\integration\orabpel\samples\references\Catch`

Compensate Activity

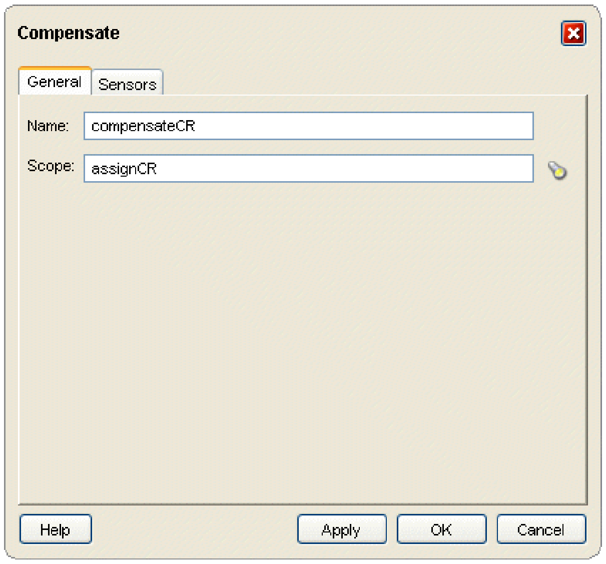
This activity invokes compensation on an inner **scope** activity that has already successfully completed. This activity can be invoked only from within a fault handler or another compensation handler. Compensation occurs when a process cannot complete several operations after already completing others. 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 C–4 appears. You can perform the following tasks:

- Click the **General** tab to provide the **compensate** activity with a meaningful name and select the **scope** activity on which the compensation handler is to be invoked.

Figure C–4 Compensate Activity



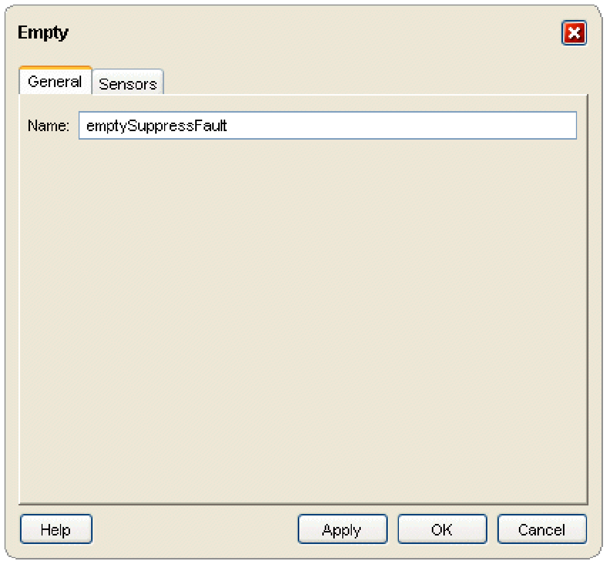
Name: compensateCR

Scope: assignCR

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 C–5 shows the **empty** activity.

Figure C–5 Empty Activity



Name: emptySuppressFault

Flow Activity

A **flow** 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 C–6 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 C–7.

Figure C–6 Flow Activity (At Time of Creation)

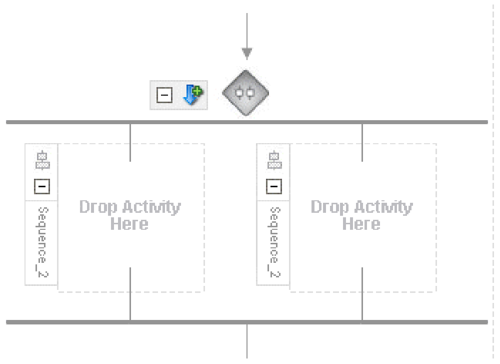
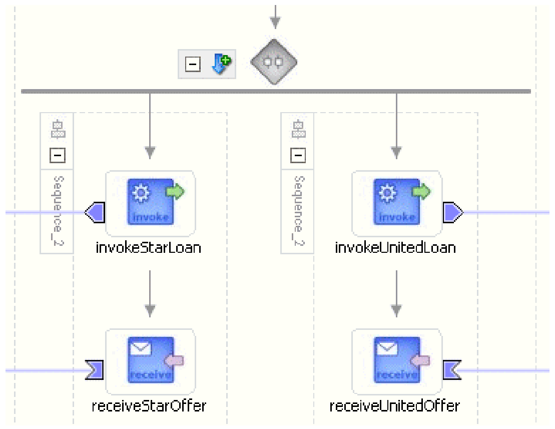


Figure C–7 Flow Activity (After Design Completion)



See Also: The following documentation for examples of using the **flow** activity:

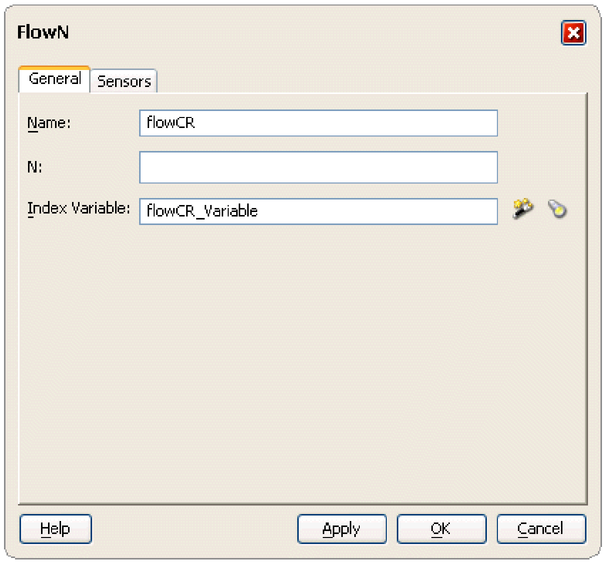
- *Oracle BPEL Process Manager Quick Start Guide*
- *Oracle BPEL Process Manager Order Booking Tutorial*
- `Oracle_Home\integration\orabpel\samples\references\Flow`

FlowN Activity

This activity enables you to create activities within a flow. You specify the number of branches of these activities to create.

Figure C–8 shows a **flowN** activity.

Figure C–8 FlowN Activity



Name: flowCR

N:

Index Variable: flowCR_Variable

See Also: *`Oracle_Home`*`\integration\orabpel\samples\references\FlowN`

Invoke Activity

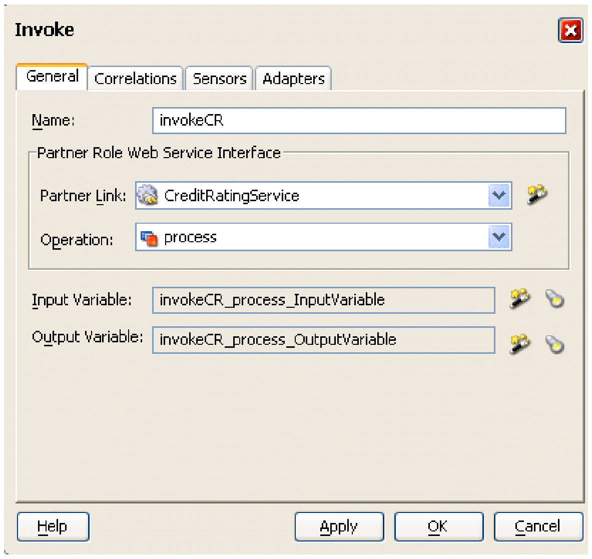
This activity enables you to specify an operation you want to invoke for the service (identified by its partner link). The operation can be one-way or request-response on a port provided by the service. You can also automatically create variables in an **invoke** activity. An **invoke** activity invokes 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 C–9 appears. You can perform the following tasks:

- Provide the **invoke** 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 C–9 Invoke Activity



Name: invokeCR

Partner Role Web Service Interface

Partner Link: CreditRatingService

Operation: process

Input Variable: invokeCR_process_InputVariable

Output Variable: invokeCR_process_OutputVariable

See Also: The following documentation for many examples of using the **invoke** activity:

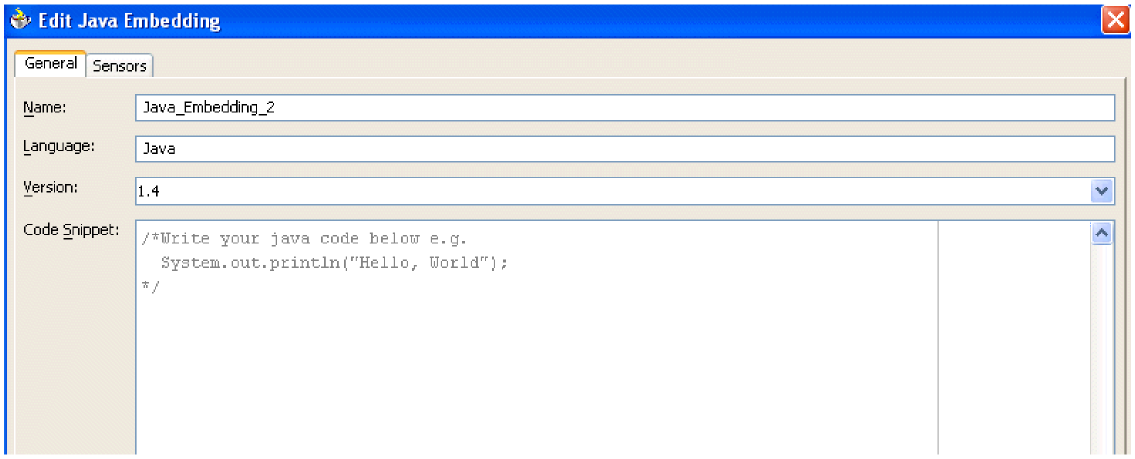
- *Oracle BPEL Process Manager Quick Start Guide*
- *Oracle BPEL Process Manager Order Booking Tutorial*
- `Oracle_Home\integration\orabpel\samples\references\Invoke`

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 C–10 appears.

Figure C–10 Java Embedding Activity



Name: Java_Embedding_2

Language: Java

Version: 1.4

Code Snippet:

```
/*Write your java code below e.g.
  System.out.println("Hello, World");
*/
```

Note: If you use this activity, ensure that you add the JAR files to the Oracle JDeveloper classpath or put them in the `Oracle_Home\integration\jdev\jdev\lib\ext` directory to ensure that your project compiles properly.

Notification Activity

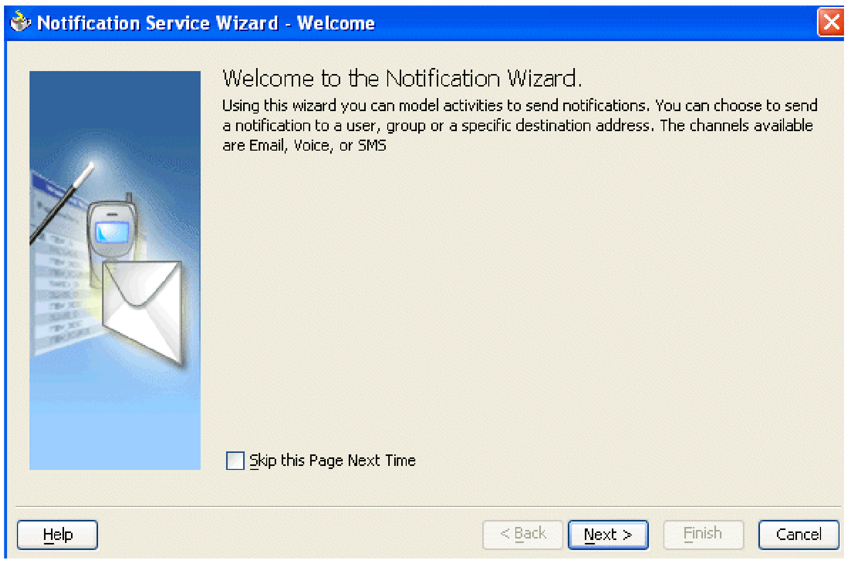
This activity enables you to send notification about an event to a user, group, or destination address. You can send a notification by e-mail, voice mail, or short message service (SMS).

For example, an Online Shopping business process of an online bookstore sends a courtesy notification 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 notification address (e-mail address in this case) from Oracle Internet Directory and sends the message to your e-mail address.

When you drag and drop a **notification** activity into JDeveloper BPEL Designer, a Notification wizard starts and prompts you for the above information.

Figure C–11 shows the Notification wizard.

Figure C–11 Notification Activity


Notification Service Wizard - Welcome

Welcome to the Notification Wizard.

Using this wizard you can model activities to send notifications. You can choose to send a notification to a user, group or a specific destination address. The channels available are Email, Voice, or SMS

See Also:

- *Oracle BPEL Process Manager Order Booking Tutorial* for an example of using the **notification** activity
- Chapter 15, "Oracle BPEL Process Manager Notification Service"

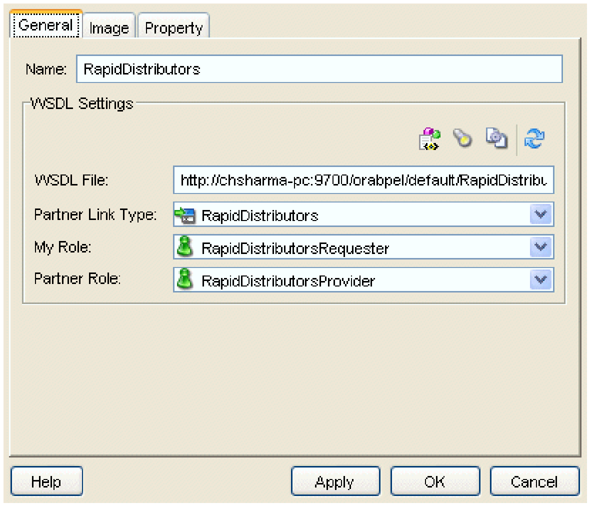
PartnerLink Activity

This activity 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 C–12 appears. You can 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 process requesting the service (defined as **My Role**)
- The role of the service (defined as **Partner Role**)

Figure C–12 PartnerLink Activity



Name: RapidDistributors

WSDL File: http://chsharma-pc:9700/orabpel/default/RapidDistribu

Partner Link Type: RapidDistributors

My Role: RapidDistributorsRequester

Partner Role: RapidDistributorsProvider

See Also: The following documentation for many examples of using the **PartnerLink** activity:

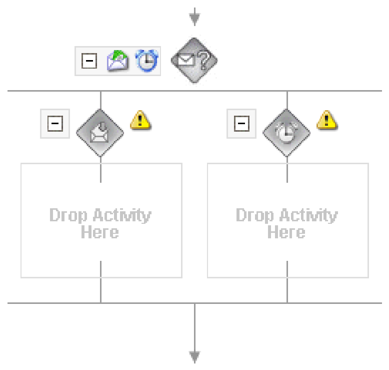
- *Oracle BPEL Process Manager Order Booking Tutorial*
- *Oracle BPEL Process Manager Quick Start Guide*

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 C–13 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.

Figure C–13 Pick Activity



See Also: `Oracle_Home\integration\orabpel\samples\references\Pick` for an example of using the **pick** activity

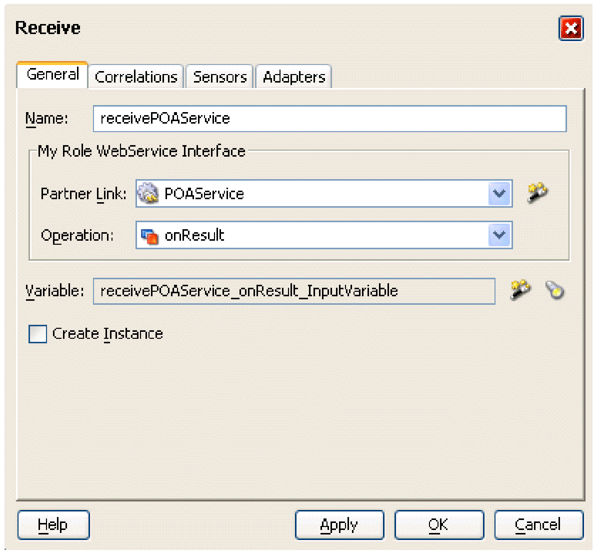
Receive Activity

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 C–14 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

Figure C–14 Receive Activity



Name: receivePOAService

My Role WebService Interface

Partner Link: POAService

Operation: onResult

Variable: receivePOAService_onResult_InputVariable

Create Instance

See Also: The following documentation for many examples of using the **receive** activity:

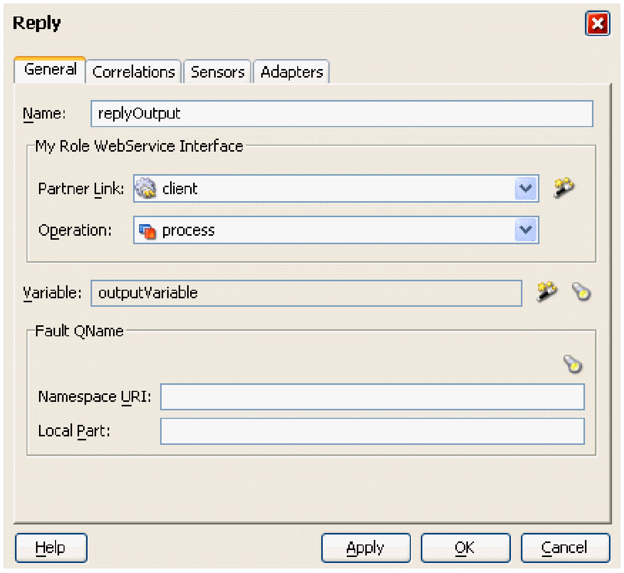
- *Oracle BPEL Process Manager Order Booking Tutorial*
- *Oracle BPEL Process Manager Quick Start Guide*
- `Oracle_Home\integration\orabpel\samples\references\Receive` (*Oracle_Home* in italics)

Reply 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 C–15 shows the **reply** activity.

Figure C–15 Reply Activity


Reply

General | Correlations | Sensors | Adapters

Name: replyOutput

My Role WebService Interface

Partner Link: client

Operation: process

Variable: outputVariable

Fault QName

Namespace URI:

Local Part:

See Also: `Oracle_Home\integration\orabpel\samples\references\Reply` for an example of using the **reply** 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:

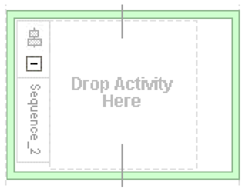
- A customer request is received in a **receive** activity.
- The request is processed inside a **flow** activity that enables concurrent behavior.
- A reply message with the final approval status of the request is sent back to the customer in a **reply** activity.

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 **sequence** activity shown in Figure C–16 appears. Define appropriate activities inside the **sequence** activity.

Figure C–16 Sequence Activity



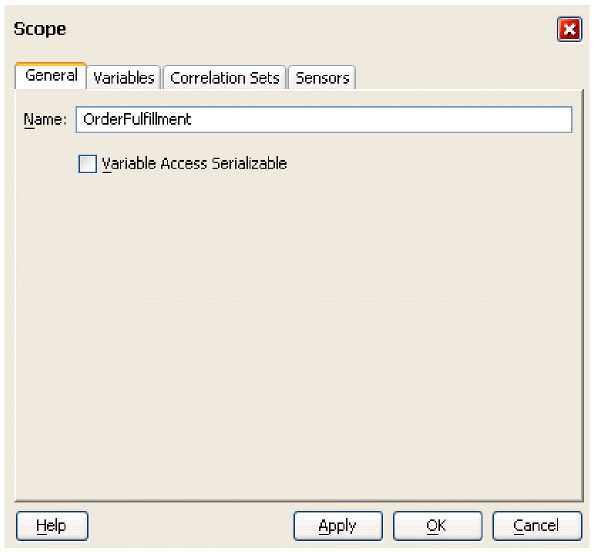
Scope Activity

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 C–17 appears. Define appropriate activities inside the **scope** activity.

Figure C–17 Scope Activity



Name: OrderFulfillment

☐ Variable Access Serializable

See Also: The following documentation for many examples of using the **scope** activity:

- *Oracle BPEL Process Manager Order Booking Tutorial*
- *Oracle BPEL Process Manager Quick Start Guide*

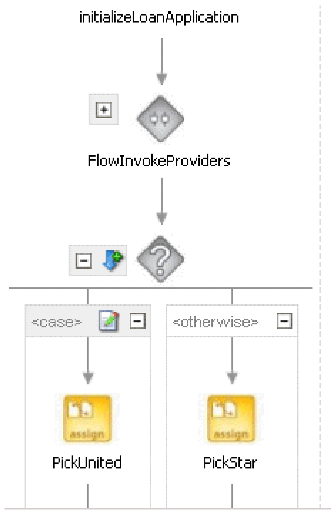
Switch Activity

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 C–18 shows a **switch** activity with two defined branches.

Figure C–18   Switch Activity



See Also: The following documentation for examples of using the **switch** activity:

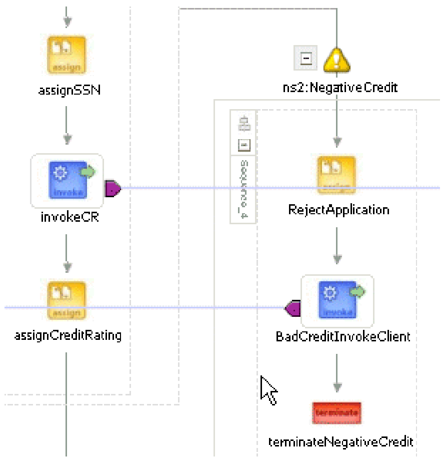
- *Oracle BPEL Process Manager Order Booking Tutorial*
- *Oracle BPEL Process Manager Quick Start Guide*
- `*Oracle_Home*\integration\orabpel\samples\references\Switch`

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 C–19 shows a **terminate** activity at the end of a **catch** branch of a **scope** activity.

Figure C–19   Terminate Activity



See Also: The following documentation for examples of using the **terminate** activity:

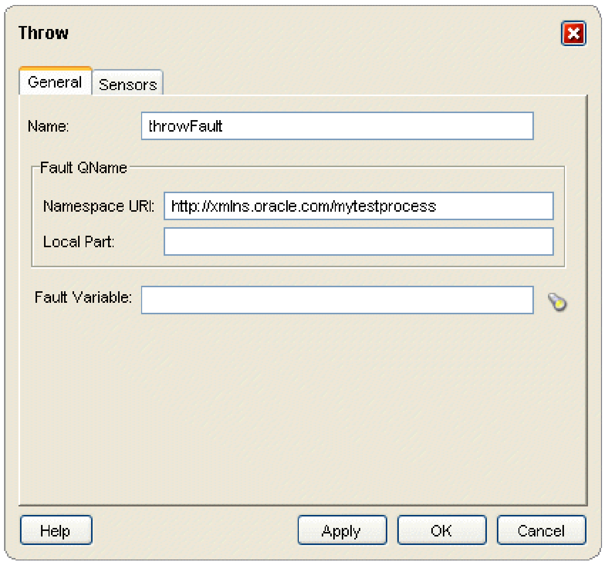
- *Oracle BPEL Process Manager Order Booking Tutorial*
- *Oracle BPEL Process Manager Quick Start Guide*
- `Oracle_Home\integration\orabpel\samples\references\Terminate`

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 C–20 appears.

Figure C–20 Throw Activity



Name: throwFault

Fault QName

Namespace URI: http://xmlns.oracle.com/mytestprocess

Local Part:

Fault Variable:

See Also: `Oracle_Home\integration\orabpel\samples\references\Throw` for an example of using the **throw** activity

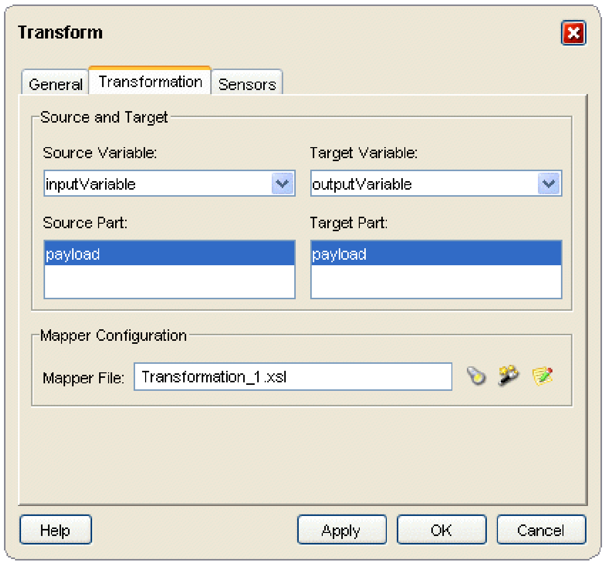
Transform Activity

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 C–21 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.

Figure C–21 Transform Activity


Transform

General | Transformation | Sensors

Source and Target

Source Variable: inputVariable

Target Variable: outputVariable

Source Part: payload

Target Part: payload

Mapper Configuration

Mapper File: Transformation_1.xsl

See Also: The following documentation for examples of using the **transform** activity:

- *Oracle BPEL Process Manager Order Booking Tutorial*
- `Oracle_Home\integration\orabpel\samples\demos\XSLMapper` (with *Oracle_Home* in italics)

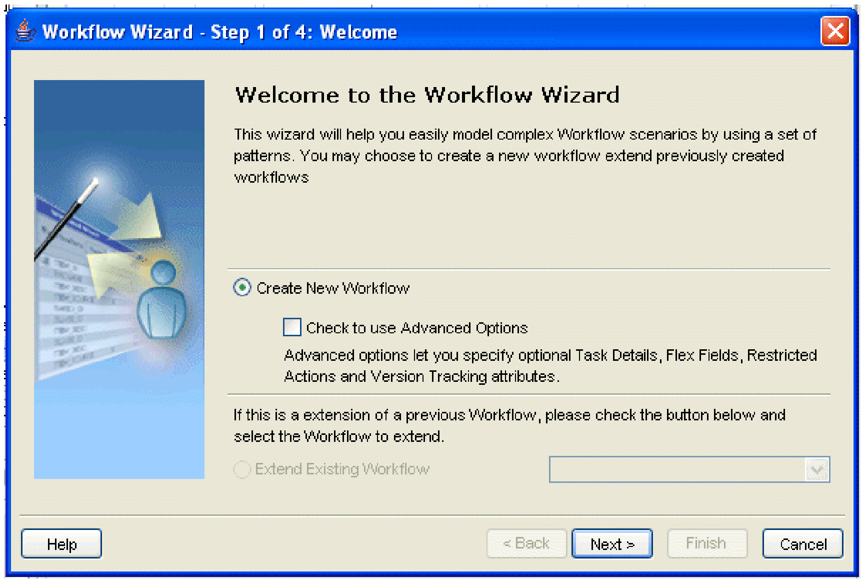
User Task

This activity enables you to start the Workflow wizard. A workflow describes the tasks, input or output information, and procedural steps that must be 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 online form and successfully completes their step before enabling the process to proceed to the next person and procedural step.

This wizard enables you to create a workflow model to manage and enforce the consistent handling of work. After wizard creation, you can also further customize a workflow pattern by directly modifying the configuration files. 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 user task, the Welcome window shown in Figure C–22 appears.

Figure C–22 User Task Activity



Welcome to the Workflow Wizard

This wizard will help you easily model complex Workflow scenarios by using a set of patterns. You may choose to create a new workflow extend previously created workflows

(●) Create New Workflow

[ ] Check to use Advanced Options

Advanced options let you specify optional Task Details, Flex Fields, Restricted Actions and Version Tracking attributes.

If this is a extension of a previous Workflow, please check the button below and select the Workflow to extend.

( ) Extend Existing Workflow

See Also: The following documentation for examples of using the **user task** activity and workflows:

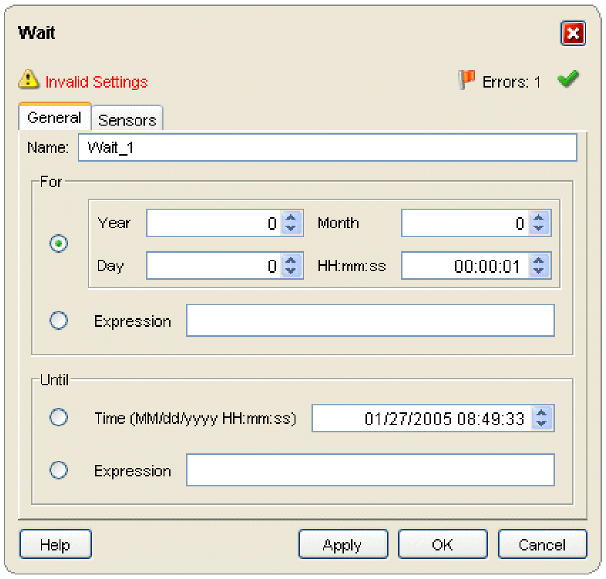
- *Oracle BPEL Process Manager Order Booking Tutorial*
- `*Oracle_Home*\integration\orabpel\samples\demos\HelpDeskServiceRequest`
- `*Oracle_Home*\integration\orabpel\samples\demos\TimeOffRequestDemo`
- `*Oracle_Home*\integration\orabpel\samples\demos\VacationRequest`
- Chapter 16, "Oracle BPEL Process Manager Workflow Services"
- Chapter 17, "Worklist Application"

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 C–23 appears.

Figure C–23 Wait Activity



Name: Wait_1

For:
- [✓] Year 0, Month 0, Day 0, HH:mm:ss 00:00:01
- Expression

Until:
- Time (MM/dd/yyyy HH:mm:ss) 01/27/2005 08:49:33
- Expression

See Also: The following documentation for examples of using the **wait** activity:

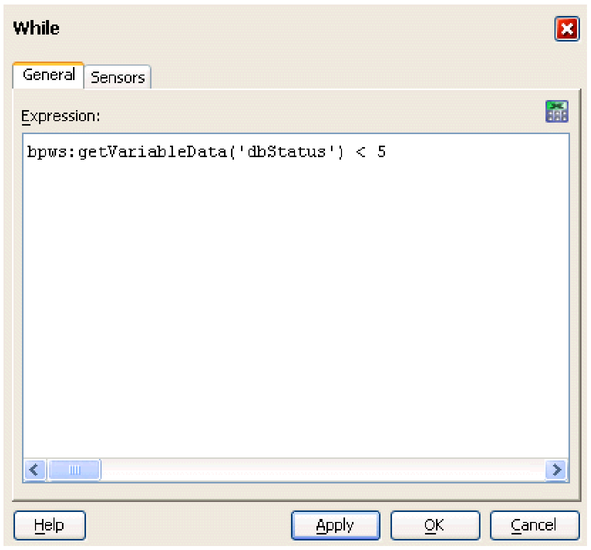
- *Oracle BPEL Process Manager Order Booking Tutorial*
- *Oracle_Home*`\integration\orabpel\samples\references\Wait`

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 C–24 appears. You can enter expressions in this window.

Figure C–24 While Activity



Expression:

```
bpws:getVariableData('dbStatus') < 5
```

See Also: The following documentation for examples of using the **while** activity:

- *Oracle BPEL Process Manager Order Booking Tutorial*
- `Oracle_Home\integration\orabpel\samples\references\While`

Summary

This appendix describes the activities you can drag and drop into a BPEL process with JDeveloper BPEL Designer.

D
User Task 2.0 Macro

This appendix describes the user task 2.0 macro.

This appendix contains the following topics:

- Introduction to User Task 2.0 Macro
- BPEL User Task Use Case
- The TaskManager Service
- Additional Capabilities of the TaskManager Service
- Summary

Introduction to User Task 2.0 Macro

The user task 2.0 macro supports user tasks from release 2.0. The user task 2.0 macro is available for backward compatibility and is replaced in this release. See Chapter 16, "Oracle BPEL Process Manager Workflow Services" to learn about the new system for handling user tasks.

BPEL processes compose multiple services into one process flow. There are frequently tasks in that flow that require user input. For example:

- A customer fills out a form
- A loan offer approves an application
- An error appears that requires a human decision to resolve

This appendix discusses the tools available for adding user tasks to a BPEL process, as well as designing interfaces to collect data from users.

BPEL User Task Use Case

This appendix uses an example of a loan officer reviewing a loan application and then approving or denying it.

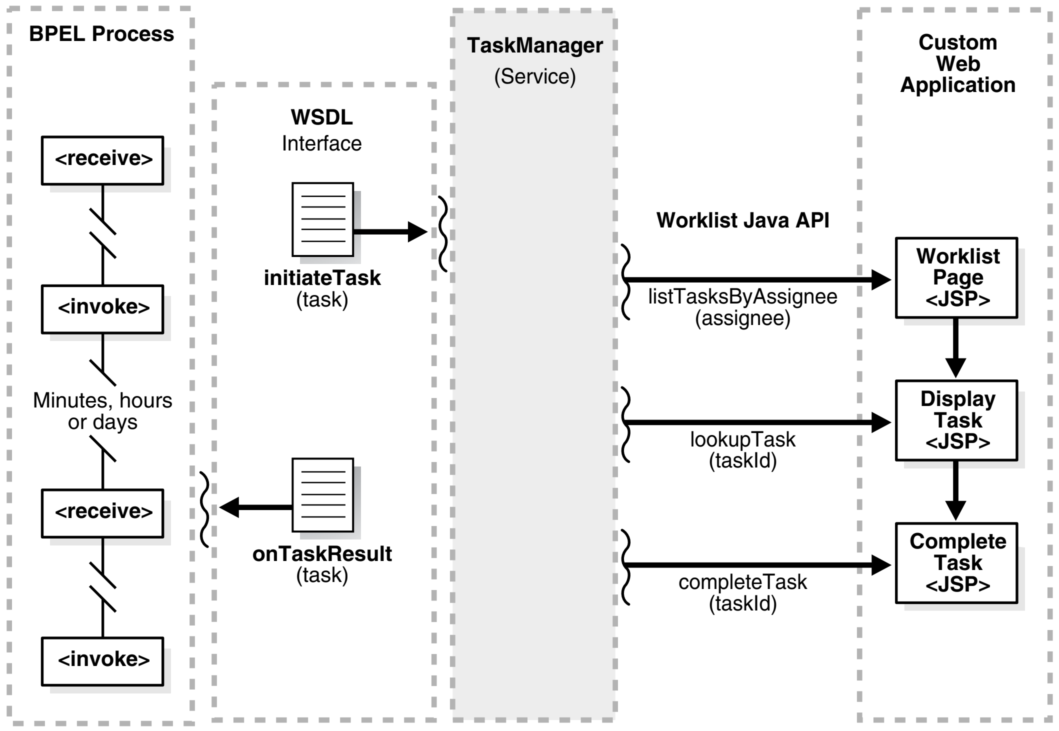
See Also: The following samples:

- `*Oracle_Home*\integration\orabpel\samples\tutorials\110.UserTasks` (for JDeveloper BPEL Designer)
- `C:\orabpel\samples\tutorials\110.UserTasks` (for Eclipse BPEL Designer)

The TaskManager Service

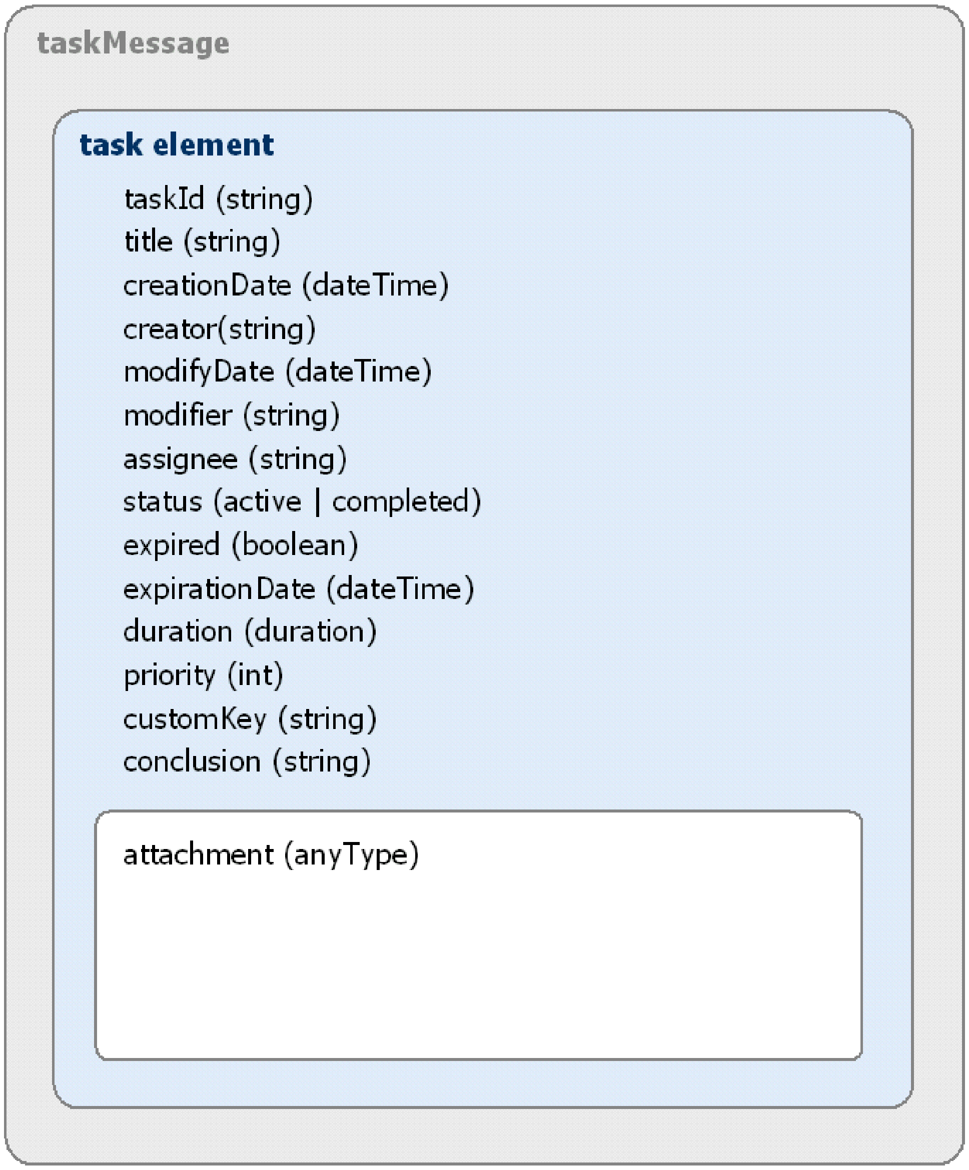
Oracle BPEL Process Manager provides a TaskManager service to help model user interactions with the BPEL process. The asynchronous TaskManager service has a WSDL interface so that a BPEL process can initiate a user task by calling the TaskManager, and receive a response, just as with any other Web service. The TaskManager has a Java Worklist API that you can use to build graphical user interface applications that show users which information is needed and return user input to the TaskManager (and then on to the BPEL process). The WSDL interface also allows the BPEL process to update a task in process, for example, to assign it to a different user if the first user is unavailable. Figure D–1 provides an overview.

Figure D–1 TaskManager Service



A task document includes all information about a task. The task document in wrapped in a taskMessage, and all inbound and outbound operations use the same taskMessage to exchange information about the task. Figure D–2 provides an overview.

Figure D–2 taskMessage



`Task.xsd` at `http://localhost:9700/orabpel/default/TaskManager/Task.xsd` is a sample task document.

```
<xs:element name="task">
 <xs:complexType>
   <xs:sequence>
     <xs:element name="taskId" type="xs:string" minOccurs="0"/>
     <xs:element name="title" type="xs:string" minOccurs="0"/>
     <xs:element name="creationDate" type="xs:dateTime" minOccurs="0"/>
     <xs:element name="creator" type="xs:string" minOccurs="0"/>
     <xs:element name="modifyDate" type="xs:dateTime" minOccurs="0"/>
     <xs:element name="modifier" type="xs:string" minOccurs="0"/>
     <xs:element name="assignee" type="xs:string" minOccurs="0"/>
     <xs:element name="status" minOccurs="0">
       <xs:simpleType>
         <xs:restriction base="xs:string">
```

```
              <xs:enumeration value="active"/>
              <xs:enumeration value="completed"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
        <xs:element name="expired" type="xs:boolean" minOccurs="0"/>
        <xs:element name="expirationDate" type="xs:dateTime" minOccurs="0"/>
        <xs:element name="duration" type="xs:duration" minOccurs="0"/>
        <xs:element name="priority" type="xs:int" minOccurs="0"/>
        <xs:element name="template" type="xs:string" minOccurs="0"/>
        <xs:element name="customKey" type="xs:string" minOccurs="0"/>
        <xs:element name="conclusion" type="xs:string" minOccurs="0"/>
        <xs:element name="attachment" type="xs:anyType"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
```

Table D–1 describes the fields in the task document.

Table D–1 Task Document Fields

Field	Description
`taskId(string)`	ID used to identify a task; automatically set by the TaskManager service when the task is created. **See Also**: `customKey (string)`
`title(string)`	Typically used when you are presented with a list of the tasks you need to complete, for example, `Approval for order #223`.
`creationDate (dateTime)`	Automatically set by the TaskManager service when the task is created.
`creator (string)`	ID of the application, system, or (sometimes) user initiating the task; typically used by the graphical user interface to partition the complete worklist into categories. When the BPEL designer (Eclipse BPEL Designer or JDeveloper BPEL Designer) is used to integrate the TaskManager service into a process, it sets this value to the name of the process initiating the task.
`modifyDate (dateTime)`	Managed by the TaskManager service; defines when the task was last modified.
`modifier (string)`	ID of the user or role performing the task update or completion operation. The semantics of the ID are owned by the application.
`assignee (string)`	ID of the user, role, or group responsible for completing the task. The semantics of the ID are opaque to both Oracle BPEL Process Manager and the TaskManager service: the BPEL process sets the assignee ID, and the graphical user interface queries the list of tasks, passing in the ID.
`status (active | completed)`	Managed by the TaskManager service; equal to `active` or `completed`.
`expired (boolean)`	Managed by the TaskManager service; indicates whether or not the task has expired.
`expirationDate (dateTime)`	Optional; defines when the task expires. **See Also**: `duration (duration)`
`duration (duration)`	Optional; the duration after which the task expires. When both an `expirationDate` and a `duration` are provided, the `expirationDate` prevails.

Table D–1 (Cont.) Task Document Fields

Field	Description
`priority (int)`	Optional; an integer marking the priority of the task. The semantics are left to the BPEL process and graphical user interface application.
`customKey (string)`	Optional; an application-specific key. The BPEL process and graphical user interface application can use either `taskId` or `customKey` when looking up a specific task.
`conclusion (string)`	Optional; an application-specific field used to tell the BPEL process how the task was completed; for example, `Approved`, `Refused`, or `Canceled`. A common pattern in the BPEL process is that the step after the completion of the task is a switch activity that keys off the conclusion field. Note that this kind of information can also be passed through the attachment field.
`attachment (anyType)`	Optional; application-specific data of any type, for any purpose of the application.

Note: Correlation and callback address information is not in the taskMessage, but instead is in a WS-Addressing header.

Integrating the TaskManager Service into a BPEL Process

The general steps for integrating the TaskManager service into a BPEL process are:

1. Define a partner link for the TaskManager service.
2. Declare and initialize the task document.
3. Invoke the `initiateTask` operation of the TaskManager service just like any other standard Web service, using an invoke activity.
4. Wait for the `onTaskResult` callback from the TaskManager service, using a receive activity.
5. Read the updated task document from the callback message.

Defining a Partner Link for the TaskManager Service

You need a partner link in your process to indicate that you are calling the TaskManager service from your BPEL process (just as for any other Web service). The WSDL file for the TaskManager service can be found on your local Oracle BPEL Process Manager installation at this location:

```
http://localhost:9700/orabpel/default/TaskManager/TaskManager.wsdl
```

The outcome of adding the partner link is as illustrated in the `TaskSample` process referenced in the See Also note on page D-2. The deployment descriptor `bpel.xml` file references the TaskManager service WSDL file, as shown below.

```
<properties id="reviewManager">
   <property name="wsdlLocation">
      http://localhost:9700/orabpel/default/TaskManager/TaskManager?wsdl
   </property>
</properties>
```

The BPEL process definition in the `TaskSample.bpel` file uses this property to define a partner link for the TaskManager.

```
xmlns:task="http://services.oracle.com/bpel/task"
.
<partnerLink name="reviewManager"
             partnerLinkType="task:TaskManager"
             partnerRole="TaskManager"
                myRole="TaskManagerRequester"/>
```

Declare and Initialize the Task Document

Before the TaskManager service can be invoked, a taskMessage must be constructed. The following BPEL code from `TaskSample.bpel` shows how to construct this message using the BPEL `assign` activity.

```
<scope name="review" variableAccessSerializable="no">
   <variables>
      <variable name="reviewTask"
                element="task:task"/>
...
</variables>
<sequence>
...
<assign name="configureTask">
   <!-- Assign 'title' in task document -->
   <copy>
      <from variable="input" part="payload"
         query="/stockReviewSheet/symbol"/>
      <to variable="reviewTask" query="/task/title"/>
   </copy>
   <!-- Assign 'assignee' in task document -->
   <copy>
      <from expression="string('jsmith@finance.com')"/>
      <to variable="reviewTask" query="/task/assignee"/>
   </copy>
   <!-- ... See the full source for the other field settings -->
   <!-- Assign 'attachment' in task document -->
   <copy>
      <from variable="input" part="payload"/>
      <to variable="reviewTask" query="/task/attachment"/>
   </copy>
</assign>
```

Initiate the Task

The next step in your BPEL process is to initiate the TaskManager by invoking its `initiateTask` operation, passing the data defined above. Specifically, you pass a taskMessage that you set up as a wrapper around the task document.

```
<scope name="reviewUserInteraction" variableAccessSerializable="no">
   <variables>
      <variable name="taskRequest" messageType="task:taskMessage"/>
      ...
   </variables>
   <sequence>
      <!-- Assign task document to taskMessage -->
      <assign name="setPayload">
         <copy>
            <from variable="reviewTask"/>
            <to variable="taskRequest" part="payload"/>
         </copy>
      </assign>
   <!-- Initiate task -->
   <invoke name="initiateTask"
           partnerLink="review"
           portType="task:TaskManager"
           operation="initiateTask"
       inputVariable="taskRequest"/>
      ...
   </sequence>
</scope>
```

This creates the task and assigns it to the assignee specified in the task document. The task is available to you through the Worklist Java API.

Task Completion

At this point, the task is created and assigned to the assignee specified in the task document and is available through the Worklist Java API.

The BPEL process waits for the TaskManager service to call it back. The callback passes back an updated taskMessage, indicating that the task has been completed or has expired.

```
<variable name=" taskResponse"
   messageType="task:taskMessage"/>
   ...
<!-- Receive the outcome of the task -->
<receive name="receiveTaskResult"
         partnerLink="review"
         portType="task:TaskManagerCallback"
         operation="onTaskResult"
         variable="taskResponse"/>
<!-- Read task document from taskMessage -->
<assign name="readPayload">
   <copy>
      <from variable="taskResponse" part="payload"/>
      <to variable="reviewTask"/>
   </copy>
</assign>
```

The `receive` activity shown above does not complete until a callback is received from the TaskManager service. As with all asynchronous activities, the BPEL process is dehydrated reliably during this time, and rehydrated and executed when the task completion (or expiration) event occurs.

The type for the TaskManager service response data is the same taskMessage message type that was used for the initiate message. However, because the type for the attachment field is an XML schema `anyType` and is application-defined, the attachment data returned can be any type and specifically does not need to be the same type as the initiate message attachment.

Typically, the conclusion field contains information that tells the BPEL process how the task was completed (for example, `Approved`, `Rejected`, or `Canceled`). This kind of information can also be passed through the attachment; it is up to the programmer to pass task data in the preferred manner.

Using Eclipse BPEL Designer to Integrate the TaskManager Service

This section reviews how to integrate the TaskManager service into a BPEL process using Eclipse BPEL Designer (based on version 0.6). It assumes you are already familiar with the basics for creating an asynchronous process, as discussed in Tutorial

2, *Developing a Credit Flow BPEL Process* at
http://www.oracle.com/technology/bpel.

Eclipse BPEL Designer simplifies matters considerably for you by generating much of the code described earlier in this chapter. This enables you to drag a user task from the **BPEL Palette** into your process. The following steps describe how to add a user task named **review** to a new asynchronous BPEL process (called **TestTask**) created in Eclipse BPEL Designer; in actual practice, of course, your starting point is a process that you are developing.

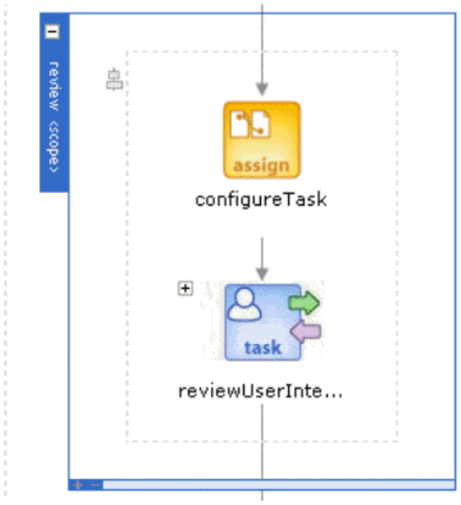
1. In the **Process Map** view of the BPEL file, drag a user task from the **BPEL Palette** (specifically, the last item in the **More Activities** list) to the transition arrow between the **initiate (client) receive** activity and the **onResult (client) invoke** callback activity.

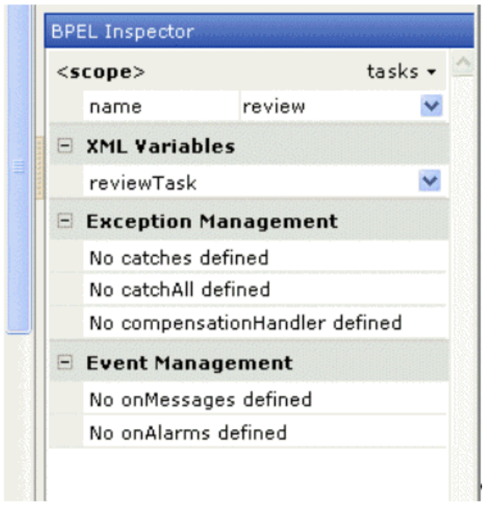
2. In the User Task window that appears, enter **review** as the task name and click **Done**.

   A **scope** activity named **review** appears in Eclipse BPEL Designer. This scope is the currently selected element, so the **BPEL Inspector** displays information about it. For example, it defines an XML variable named **reviewTask** (which is the task document).

   > **Note:** All names beginning with **review** in this example generally begin with the task name you specify in the User Task window.

3. Expand the newly created scope by clicking the **+** icon to the left of it in the **Process Map** view. Within the expanded scope, you see an **assign** activity (named **configureTask**) and a task that can be expanded further.

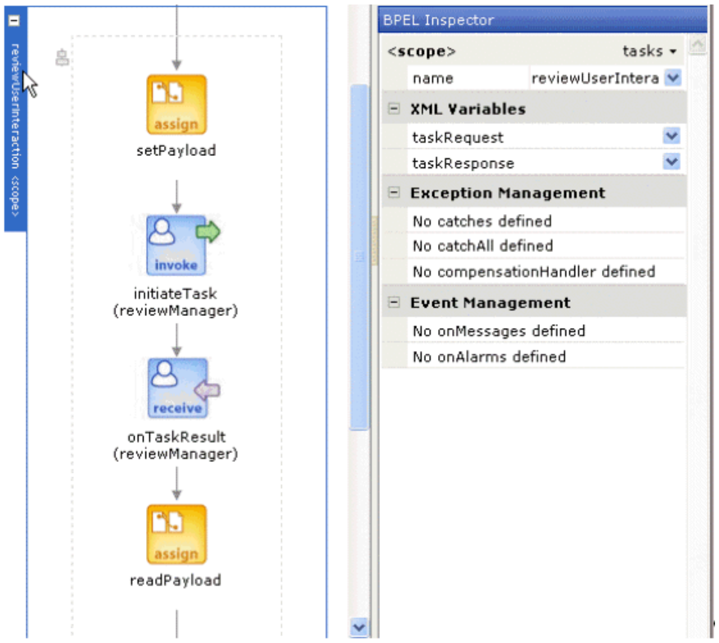



BPEL Inspector

<scope>	tasks ▾
name	review
XML Variables	
reviewTask	
Exception Management	
No catches defined	
No catchAll defined	
No compensationHandler defined	
Event Management	
No onMessages defined	
No onAlarms defined	

   The **configureTask** assignment initializes the task document; however, before going to that level, you do one more expansion, to get an overview of the process.

4. Expand the task (**reviewUserInteraction**) by clicking its **+** icon and notice that it expands into a scope containing the **invoke** and **receive** activities for the task, along with two **assign** activities. Click the label along the left edge of this scope. This selects the scope and enables you to see its contents in the **BPEL Inspector**.



BPEL Inspector

<scope> tasks

name	reviewUserIntera

XML Variables

taskRequest
taskResponse

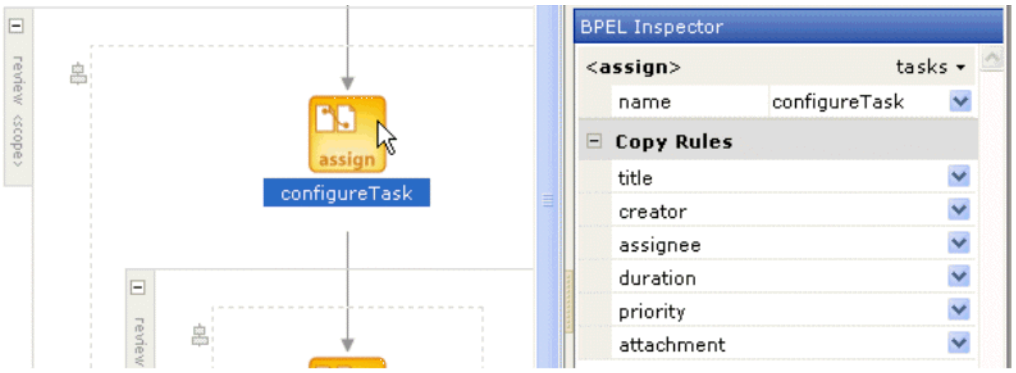
Exception Management

No catches defined
No catchAll defined
No compensationHandler defined

Event Management

No onMessages defined
No onAlarms defined

5. Scroll back in the **Process Map** if necessary to the outer scope (named **review**) and click the **configureTask** assignment within it. Listed under **Copy Rules** in the **BPEL Inspector** are the fields of the **reviewTask** task document.



BPEL Inspector

<assign> tasks

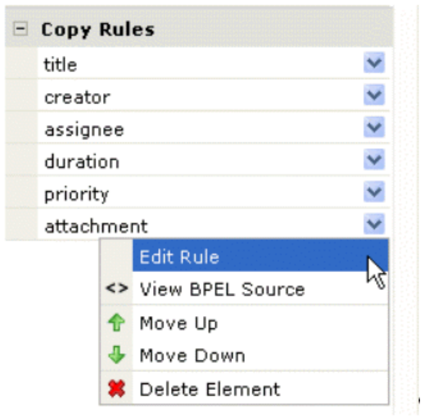
name	configureTask

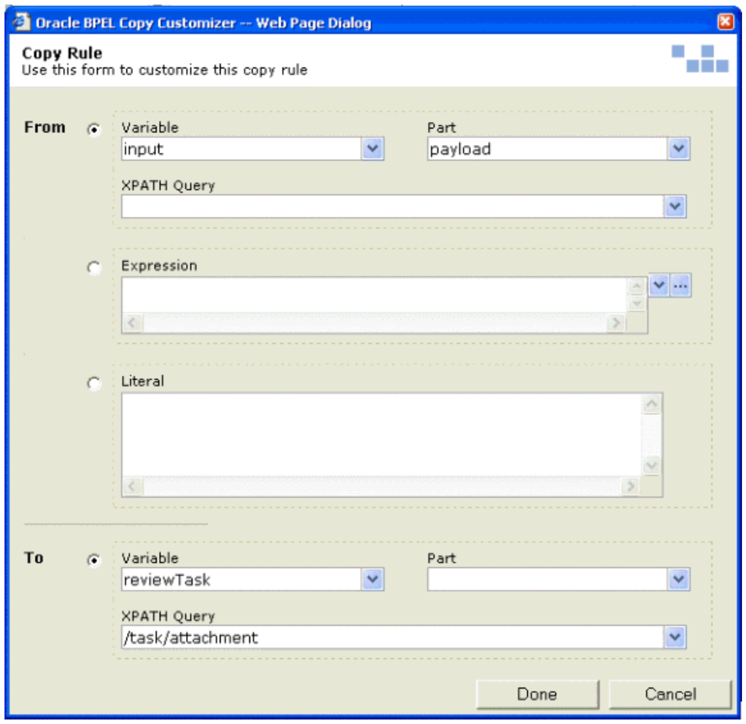
Copy Rules

title
creator
assignee
duration
priority
attachment

You can view the Copy Rule window for each field either by selecting **Edit Rule** in its drop-down list or by clicking the field name (which is a link). For every field except the last one, attachment, the Copy Rule window shows an assignment being made to that field. In actual practice, you change the values being assigned to the fields title through priority as appropriate, and optionally pass custom data to the task through the attachment field (as in the next step).

6. To pass data to the task, click **Edit Rule** in the drop-down list for the attachment field in the **BPEL Inspector** (or click **attachment**) and complete the **From** part of the Copy Rule window (for example, to pass the payload part of the input variable as described in "Declare and Initialize the Task Document" on page D-7). If you do not want to pass anything in the attachment field, you must click **Delete Element** in the field's drop-down list (or the process does not compile because of the incomplete assignment).





Copy Rule
Use this form to customize this copy rule

From (●) Variable: input Part: payload
XPATH Query:
( ) Expression
( ) Literal

To (●) Variable: reviewTask Part:
XPATH Query: /task/attachment

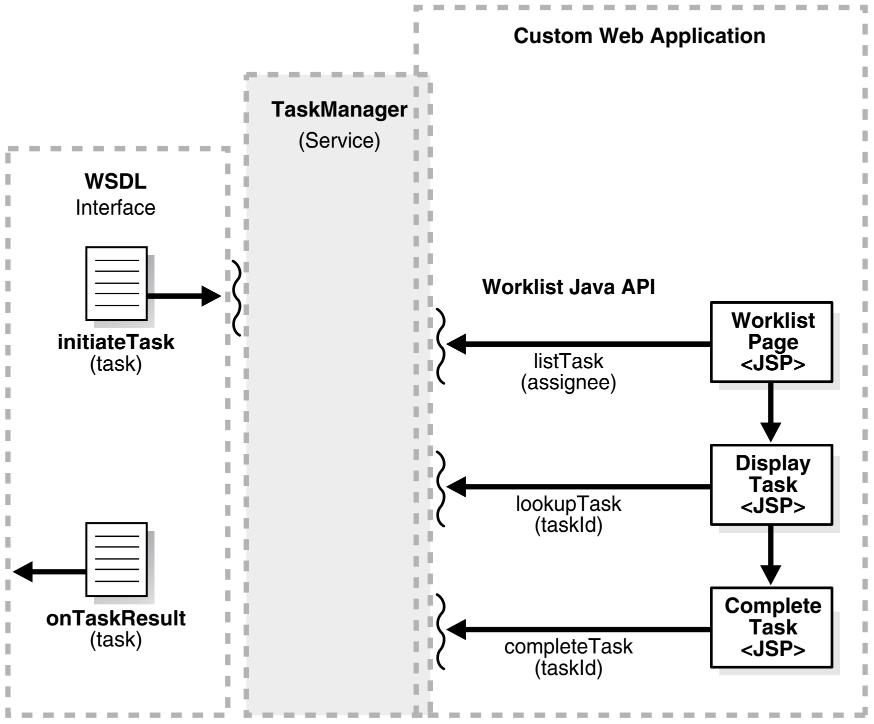
7. Click **Done** to complete the configuration of the attachment copy rule.
8. Click the next **assign** activity in the **Process Map**: **setPayload**, inside the **reviewUserInteraction** scope. You see that only **payload** is listed under Copy Rules. If you look at the Copy Rule window, **reviewTask** (the task document from the outer scope) is assigned to the **payload** part of the **taskRequest** variable as the final step before initiating the task.
9. Click the final **assign** activity (**readPayload**).
10. Explore this activity in the **BPEL Inspector** to see that this is where the task data returned in a **taskMessage** (that is, in the payload part of the **taskResponse** variable) is read and stored in the variable **reviewTask**.

Note that while the above may seem complex at first, it is due to the fact that the TaskManager service is a complex service. The **Designer User Task Palette** entry is merely a preconfigured template that encapsulates the normal patterns for use of the TaskManager service in a BPEL flow. However, Eclipse BPEL Designer continues to improve as a means of integrating the TaskManager service into a BPEL process. You also have the option of working directly in the BPEL source code.

Creating a User Interface for the Task

On the other side of a TaskManager service is typically a user interface in which the assigned users can view, update, and complete tasks. Figure D–3 provides an overview.

Figure D–3 TaskManager and a Custom Web Application



The typical steps for accessing task information from the user side are:

1. A graphical user interface uses the Java Worklist API to list the tasks assigned to a user or role.
2. The user selects a task and reviews its detail information.
3. The user updates any editable data associated with the task and saves or completes the task.

List the Assigned Tasks

Using the Java Worklist API to the TaskManager service from the client perspective involves the following high-level steps. The code shown in this section is taken from `C:\orabpel\samples\tutorials\110.UserTasks\TaskSampleUI\listTasks.jsp`.

1. Use the `com.oracle.bpel.client.Locator` class to return a worklist service handle (an `IWorklistService`).

```
<%@page import="com.oracle.bpel.client.Locator" %>
<%@page import="com.oracle.bpel.services.task.IWorklistService" %>
    ...
    // Obtain a reference to BPEL domain 'default' using password 'bpel'
```

```
      Locator locator=new Locator("default", "bpel");
      IWorklistService worklist =
        (IWorklistService)
            locator.lookupService(IWorklistService.SERVICE_NAME);
```

2. Use the `com.oracle.bpel.services.task.IWorklistService` interface to fetch `com.oracle.bpel.services.task.ITask` objects, manipulate them, complete them, and so on.

```
<%@page import="com.oracle.bpel.services.task.ITask" %>
   ...
   ITask[] tasks=
      worklist.listTasksByAssignee("jsmith@finance.com");
   ...
```

You can fetch tasks by other criteria as well. For example, you can look up a task by its unique `taskId` assigned by the TaskManager service or based on a specified creator attribute (which is programmer-defined). You can also build an arbitrary search criterion using the `com.oracle.bpel.client.util.WhereCondition` utility class (for example, to fetch all the expired tasks assigned to a particular user). For more information, including full documentation on the interfaces for the classes used in the code in this section, see Oracle BPEL Process Manager API Javadocs (located at `C:\orabpel\docs\apidocs\index.html`).

The code below shows how to iterate through the list of tasks in the JSP page and, for each task, display some basic descriptive information and a link that displays a details page (`displayTask.jsp`) for the task.

```
<h1>My Custom Task List Page</h1>
<table border="1">
<tr>
   <th>Title</th>
   <th>Due Date</th>
   <th>Priority</th>
</tr>
<%
   // Iterate through the list of tasks
   for (int i=0; i < tasks.length; i ++)
   {
      ITask thisTask=tasks[i];
      // We are interested in tasks assigned to "jsmith@finance.com"
      // and generated by the TaskSample BPEL process. Often the BPEL
      // process initiating a task will assign its name to the
      // creator field of the task document. This is at least the
      // case with the TaskSample BPEL process.
      if (! "TaskSample".equals(thisTask.getCreator()))
   {
      // This task has been generated by another BPEL process
      // and should therefore not be in the list.
      continue;
      }
      // Read the title assigned to the task. The title is to
      // a task what a subject would be to an email.
      String title=thisTask.getTitle();

      // Get the unique ID/key associated with this task.
      // This is passed from page to page to identify the
      // task the user is reviewing and completing.
      String taskId=thisTask.getTaskId();
```

```
         // Read the expiration date assigned to the task.
         Date expiration=null;
      if (thisTask.getExpirationDate() != null)
         expiration=thisTask.getExpirationDate().getTime();
      // Read priority associated with the task
      int priority=thisTask.getPriority();
%>
   <tr>
      <td>
      <a href="displayTask.jsp?taskId=<%= taskId %>">
         <%= title %></a>
      </td>
      <td>
      <%= expiration %>
      </td>
      <td>
      <%= priority %>
      </td>
   </tr>
<%
   }
%>
</table>
```

Display the Payload Data for a Task

The `displayTask.jsp` page invoked from the hyperlink in the code above gets passed a `taskId` (again, a unique string identifier generated by the TaskManager service) and uses the same `Locator` and `IWorklistService` objects as above to fetch the task associated with that `taskId`. Then, to manipulate the payload data associated with a task, it uses the facade capability, which provides a thin Java interface on top of arbitrary XML data. Note that facades are not at all like JAXB-style bindings because they do not attempt to fully map XML to Java. They merely provide a Java envelope for XML data that is then accessed through JavaBeans-style getter and setter functions. This enables facades to avoid all of the problems associated with JAXB bindings as schemas get complex.

The facade construct is described in more detail in "XML Facade" on page 10-3; however, at a very high level, the basic steps for working with the XML facade are as follows:

1. Use the `schemac` Ant task (or command-line tool) in your `build.xml` script to generate facade classes for XML schema types. For example:

   ```
   <schemac input="${basedir}/TaskSample.wsdl"
            out="${basedir}/TaskSampleUI/WEB-INF/classes"/>
   ```

   This command generates the facade classes for the XML schema types described in `TaskSample.wsdl`, namely:

   ```
   <element name="stockReviewSheet" type="finance:StockReviewSheetType" />
   <complexType name="StockReviewSheetType">
      <sequence>
         <element name="symbol" type="string" />
         <element name="targetPrice" type="float" />
         <element name="currentPrice" type="float" />
         <element name="action" type="string" />
         <element name="quantity" type="int" />
         </sequence>
   </complexType>
   ```

The facade classes generated include a factory for creating new instances of the `StockReviewSheet` element and then `getter` and `setter` methods for manipulating the data fields within the instances. If you want to review the exact interfaces exposed by the generated classes, you can either view the generated Java classes or create Javadocs for them by using the `-j` option to the `schemac` command.

2. Within your code, you can use the generated classes to manipulate instances of XML data types through a simple Java interface. The code below from `C:\orabpel\samples\tutorials\110.UserTasks\TaskSampleUI\displayTask.jsp` uses the `Locator` and `IWorklistService` objects to fetch a particular task by `taskId`. It then gets the payload of the task, which is a W3C DOM element type, and uses that and the facade `StockReviewSheetFactory` class to instantiate a `StockReviewSheet` facade instance. Finally, the facade class `getter` and `setter` methods are used to fetch appropriate data fields from the payload and display them in the JSP page.

```
<%@ page import="com.otn.samples.finance.StockReviewSheet" %>
<%@ page import="com.otn.samples.finance.StockReviewSheetFactory" %>...

Locator locator=new Locator("default", "bpel");

IWorklistService worklist=(IWorklistService)
   locator.lookupService(IWorklistService.SERVICE_NAME);

ITask task=worklist.lookupTask(taskId);

// Get a reference to the XML StockReviewSheet document attached to the
// task. If you look at the implementation of the TaskSample BPEL
// process, you will notice that a StockReviewSheet element is assigned
// to the task as an attachment.
Element rsElement=(Element) task.getAttachment();

StockReviewSheet srs =
   StockReviewSheetFactory.createFacade(rsElement);
// Use the friendly bean interface of the XML facade to access the data
// contained in the XML attachment.
String symbol=srs.getSymbol();
float targetPrice=srs.getTargetPrice();
float currentPrice=srs.getCurrentPrice();
...
<tr>
   <td>Symbol</td>
   <td><input type="text" name="Symbol"
              value="<%= symbol %>"/></td>
   <td>(String)</td>
</tr>
...
```

The `displayTask` JSP page provides an HTML form that enables the user to fill in the action and quantity fields for the stock review sheet, as well as to edit any of its other fields. When the form is submitted, it passes these values, along with the `taskId`, to `completeTask.jsp`.

Update the Payload Data and Complete the Task

The final step is to use the generated facade classes to create the appropriate attachment data element (the task document) to be returned to the task and then pass it to the TaskManager service along with an indication that the task has been completed.

In this example, this is done in `completeTask.jsp`, which creates a new instance of the `StockReviewSheet` element and fills it in with the form data passed to it. It then looks up the task by `taskId`, as before. What is new about this code from `completeTask.jsp` is shown below. It sets some fields of the task object (including the updated attachment) and then informs the worklist service that the task has been completed.

```
StockReviewSheet srs=StockReviewSheetFactory.createFacade();

// Populate symbol from data submitted as part of the post
srs.setSymbol(request.getParameter("symbol"));
..
ITask task=worklist.lookupTask(taskId);

task.setAttachment(srs.getRootElement());

// Conclusion is user-defined - here it is "buy" or "sell"
task.setConclusion(strAction);

worklist.completeTask(task);

out.println("This task has been successfully completed.");
```

At this point, the TaskManager service calls back asynchronously to the process that is waiting for this task, passing it the updated task attributes and attachment data.

Additional Capabilities of the TaskManager Service

This final section looks at some additional capabilities related to working with the TaskManager service: expirations and timeouts, notification to the assignee, task reassignment, and task assignment to groups (and role resolution).

Enabling Expiration/Timeouts for Tasks

The TaskManager service has a built-in expiration and timeout capability. You can specify the timeout period for a task as a period of time (`duration` field) or a specific point in time (`expirationDate` field). In either case, when the specified time is reached and the task has not yet been completed, an expiration event is sent as a BPEL event to the process that is waiting for completion of that TaskManager service. The waiting BPEL process then uses a BPEL event handler. At that point, the process can take any desired action (including sending a reminder e-mail, reassigning the task to someone else, canceling the task, and so on).

The `duration` and `expirationDate` fields are specified as having the XML schema types `duration` and `dateTime`, respectively. For example, a duration of one hour is specified as `PT1H` (meaning a period of time of 1 hour). In addition to `H` for hours, you can also use `M` for minutes, `S` for seconds, and so on. For more information on the format of the `duration` and `dateTime` data types, see the XML schema specification:

```
http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/#duration
```

and

```
http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/#dateTime
```

First, in the BPEL source you specify a duration or expiration date, as follows:

```
<!-- Assign 'duration' in task document -->
<copy>
   <from expression="string('PT5M')"/>
   <to variable="reviewTask" query="/task/duration"/>
</copy>
```

After initiating the task, do the following:

1. Wrap the TaskManager service receive activity in a scope.
2. Add an event handler to the scope to handle `onTaskExpired` events.
3. Within that event handler, do whatever you like in response to the expiration event. The code below reassigns the task to someone else and sets a new expiration time of one more hour.

```
<invoke name="initiateTask" partnerLink="review" portType="task:TaskManager"
operation="initiateTask" inputVariable="taskRequest"/>

<!-- Receive the outcome of the task -->
<scope name="reviewTask">
     <eventHandlers>
        <onMessage partnerLink="review" portType="task:TaskManagerCallback"
         operation="onTaskExpired" variable="taskRequest">
           <sequence>
              <assign name="reassignTask">
                <!-- Assign NEW 'assignee' in task document -->
                   <copy>
                      <from expression= "string('director@oracle.com')"/>
                      <to variable="taskRequest" part="payload"
                       query="/task/assignee"/>
                   </copy>
                <!-- Assign NEW 'duration' in task document -->
                   <copy>
                      <from expression="string('PT1H')"/>
                      <to variable="taskRequest" part="payload"
                       query="/task/duration"/>
                   </copy>
                   <copy>
                      <from expression="string('Escalated')"/>
                      <to variable="taskRequest" part="payload"
                       query="/task/status"/>
                   </copy>
                </assign>
       <invoke name="reassign" partnerLink="review" portType="task:TaskManager"
        operation="updateTask" inputVariable="taskRequest"/>
      </sequence>
    </onMessage>
  </eventHandlers>
 <receive name="receiveTaskResult" partnerLink="review"
 portType="task:TaskManagerCallback" operation="onTaskResult"
 variable="taskResponse"/>
</scope>
```

If you want to run the entire example, see the from the TimeOffRequestFlow sample available in `c:\orabpel\samples\demos\TimeOffRequestDemo`.

Sending Notifications

It is common to send a notification message to a user when a task is assigned to that user (or when the timeout duration is reached, when the task is reassigned, or in other situations). This notification can be done in BPEL at the same time that the TaskManager service is initiated; therefore, anything that can be done in BPEL (invoking a Web service, sending an e-mail message or a JMS message, executing some Java code, and so on) can be done to notify a user of a task-related event.

The TimeOffRequestFlow sample includes a code example of using the Oracle BPEL Process Manager Mail Service to send an e-mail notification to a user when a time-off request approval task has been assigned to that user. In this case, the e-mail includes an XML document attachment describing the time-off request and contains a link to a JSP page where the request can be approved or rejected.

Reassigning Tasks

Reassigning a task is as simple as changing the assignee field in the task document and then invoking the `updateTask` operation on the TaskManager service. A code example of this is shown in the "Enabling Expiration/Timeouts for Tasks" on page D-16. If the application requires that once a task is reassigned it cannot be completed by the original assignee, then the user interface that completes a task must first fetch the task document and check that the assignee has not changed before completing a task.

Assigning Tasks to Groups and Resolving Roles

The TaskManager service has a very simple construct for the assignee attribute of a task: it is a string identifier that is not interpreted by the TaskManager service to have any special meaning. Although simple, this construct is powerful and flexible. In particular, it allows for role resolution and the concept of group worklists to be implemented in a straightforward manner.

When a task is assigned to a group, this typically means that several users can see the task in their worklists. When one of them selects the task on which to work, it disappears from the worklists of the other users in the group. Additionally, it is common to set up a timeout period such that if the individual working on the task does not complete it within a specified time, it may revert back to group assignment and then reappear on the group worklists.

The most common way to implement this use case with the TaskManager service is as follows:

1. The task is assigned to the group identifier.
2. The worklist user interfaces for members of the group show all tasks assigned to the group.
3. When a user selects a task on which to work, it is checked to ensure it has not already been reassigned. If not, it is reassigned to the individual's user ID.
4. Optionally, notifications and timeouts can be used effectively here.

Role resolution works in much the same way. Because special interpretation is not given to the assignee for a task, the assignee can actually be a role, and resolution can happen on the user interface side. Alternatively, it is fairly simple to integrate a BPEL process with directory services such as LDAP and perform dynamic role resolution at the time the task is created.

Summary

BPEL includes a TaskManager service that acts as an intermediary between BPEL and a user interface for performing user tasks. This appendix discusses how to create a partner link to the TaskManager service, and how to build a JSP user interface to view task information and complete the task.

E Deployment Descriptor Properties

This appendix discusses 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 the deployment descriptor of a BPEL process, and which are accessed at run time by the BPEL process. You can change the value of a preference from Oracle BPEL Console at run time, without having to redeploy the BPEL process.

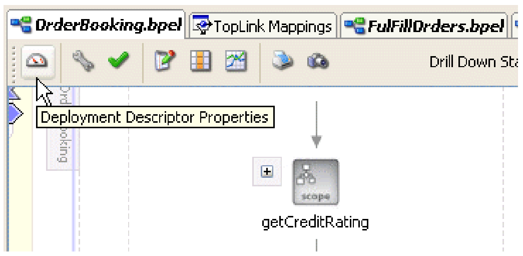
Preferences enable BPEL process designers to externalize literal values from a process, enabling them to be altered without having to redeploy the entire BPEL process.

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, the change can be performed at run time, without redeploying the process.

Defining a Preference Property

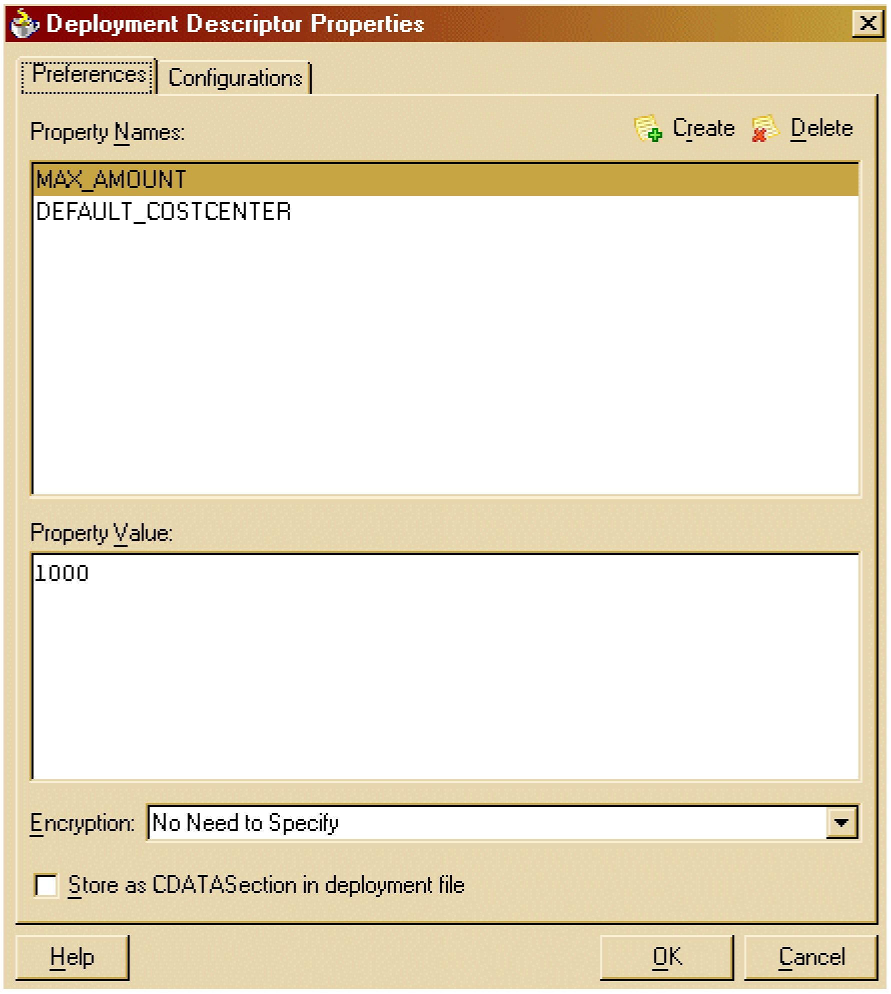
You can define preference values at run time from JDeveloper BPEL Designer. Click the **Deployment Descriptor Properties** icon shown in Figure E–1. This icon is located just above and to the left of the **Diagram View**.

Figure E–1 Deployment Descriptor Properties Icon



In the Deployment Descriptor Properties window, click the **Preferences** tab, as shown in Figure E–2.

Figure E–2 Deployment Descriptor Preference Properties in Oracle BPEL Console



Property Names:
MAX_AMOUNT
DEFAULT_COSTCENTER

Property Value:
1000

Encryption: No Need to Specify

☐ Store as CDATASection in deployment file

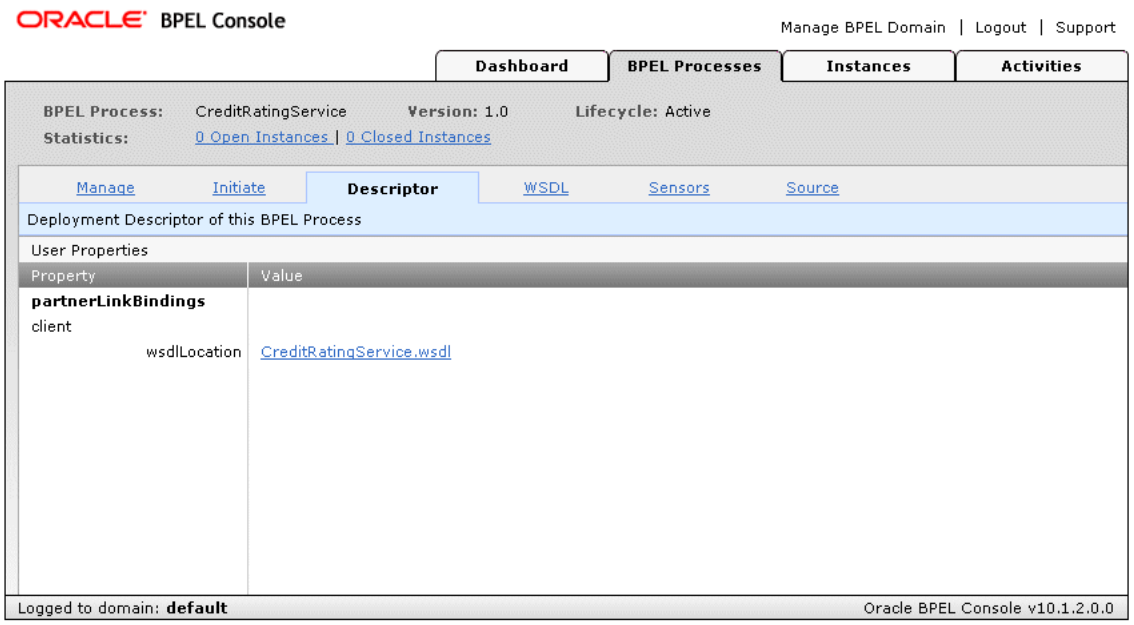
Click **Create** and enter a preference name. Edit the value in the **Property Value** field and click **OK.** The change takes effect immediately. Preferences you create in this way are reflected in `bpel.xml`, inside the `preferences` tag, as follows:

```
...
<preferences>
   <property name="MAX_AMOUNT">1000</property>
   <property name="DEFAULT_COSTCENTER">US23</property>
</preferences>
...
```

Updating a Preference at Run Time

You can update preference values at run time in Oracle BPEL Console, as shown in Figure E–3. Select the process, then select the **Descriptor** tab to display the deployment descriptor for the process, including any preferences. Update the preference value, and click **Update descriptor**. The change takes effect immediately.

Figure E–3 Updating BPEL Process Preferences at Run Time



BPEL Process: CreditRatingService Version: 1.0 Lifecycle: Active
Statistics: 0 Open Instances | 0 Closed Instances

Deployment Descriptor of this BPEL Process

User Properties

Property	Value
partnerLinkBindings	
client	
wsdlLocation	CreditRatingService.wsdl

Logged to domain: **default** Oracle BPEL Console v10.1.2.0.0

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 JDeveloper BPEL Designer. The contents are encrypted at deployment only. The **Encryption** field (see Figure E–2 on page E-2) provides the following options:

- **No Need to Specify**—Do not encrypt the contents.
- **Plain Text**—The contents remain in plain text.
- **Encrypt**—The contents are encrypted.

The property is also shown as a password field in the **Configuration** tab of the deployment descriptor.

The following example shows the XML code without encryption set and then with encryption set.

Without encryption set:

```
...
<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.

```
...
<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:

```
...
<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 JDev project), so that the XML looks like this:

```
...
<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.

Use Case for Preference Properties

The Payment Processor demo (or PayDemo) demonstrates the use of a preference property.

See Also: The Payment Processor (PayDemo) demo at *`Oracle_Home`*`\integration\orabpel\samples\demos\PaymentProcessor`

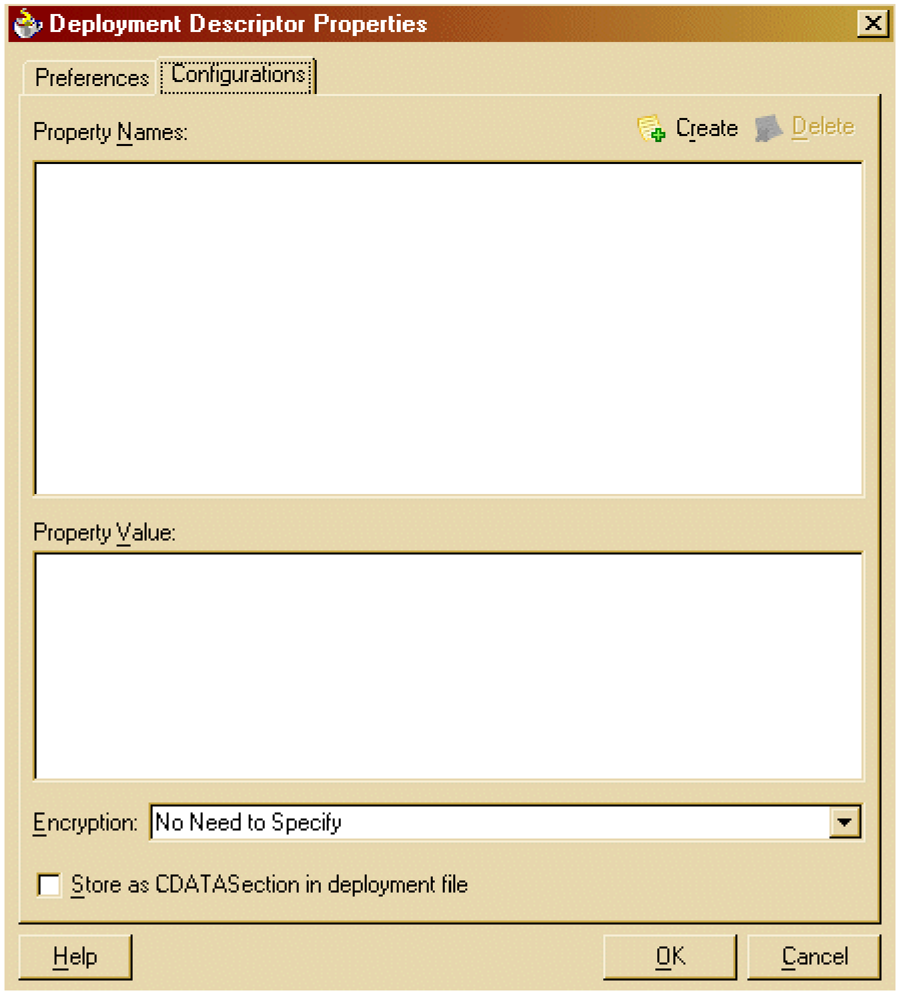
Deployment Descriptor Configuration Properties

Configuration properties are specific properties used by the server, Oracle BPEL Console, or both. In Oracle BPEL Console, for example, the configuration properties 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 at run time from JDeveloper BPEL Designer. Click the **Deployment Descriptor Properties** icon, located just above and to the left of the diagram view. (See Figure E–1 on page E-1.) In the Deployment Descriptor Properties window, click the **Configurations** tab, as shown in Figure E–4.

Figure E–4 Deployment Descriptor Configuration Properties in the Oracle BPEL Console



Click **Create** and enter a configuration name. Edit the value in the **Property Value** field and click **OK.** The change takes effect immediately. Preferences you create in this way are reflected in `bpel.xml`.

See "Encrypting a Preference Value" on page E-3 for information about encrypting the contents of configuration properties.

Table E–1 lists the property names of the `configurations` deployment descriptor. For each configuration property, a description is provided, as well as the expected behavior of the server when it is changed.

Table E–1 Configuration Properties for the configurations Deployment Descriptor

Property Name	Description	On Change
`completionPersistLevel`	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 `cube_instance` and `cube_scope` tables. The other value is `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`.	NA
`completionPersistPolicy`	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 be faulted, then only the faulted instances are saved. If this value is set to `off`, then no instances of this process are saved.	NA
`defaultInput`	The XML document that you want to use as input to test the process from Oracle BPEL Console.	Takes effect immediately
`initializeVariables`	Default value is `True`. If set to `False`, Oracle BPEL Console does not initialize the variables based on to-spec queries.	NA
`inMemoryOptimization`	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 `inMemory` optimization on the instances of this process on `to-spec` queries.	NA
`loadSchema`	Default value is `True`. If set to `False`, XML schemas are not loaded and Oracle BPEL Console becomes typeless.	NA
`noAlterWSDL`	Default value is `False`. If set to `True`, the compiler does not try to modify the process WSDL to add binding and service information.	NA
`optimizeVariableCopy`	Default value is `True`. If set to `False`, Oracle BPEL Server does not enable copy-on-write for an assign copy.	NA
`relaxTypeChecking`	Default value is `False`. If set to `True`, the compiler does not check type compatibility with an assign activity.	NA
`relaxXPathQName`	Default value is `False`. If set to `True`, the compiler throws exceptions for unqualified steps in the query. For example, where the correct form must be: `query="/ns1:payload/ns1:name"`, the following form passes compilation, if this flag is turned on: `query="/payload/name"`.	NA
`sensorActionLocation`	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.	NA
`sensorLocation`	Location of the sensor XML file. The sensor XML file defines the list of sensors into which Oracle BPEL Process Manager logs events.	NA
`testIntroduction`	Introduction text that appears in the test console.	Takes effect immediately

Table E–1 (Cont.) Configuration Properties for the configurations Deployment Descriptor

Property Name	Description	On Change
`transaction`	When set to `participate`, the process produces a fault that is not handled by fault handlers, which calls the transaction to be rolled back.	Takes effect immediately
`serviceLevelAgreement`	Service Level Agreement (Process Completion Time) - Threshold for a commitment within which a process is completed for a specified time period. Value is an XML duration.	NA
`xpathValidation`	Default value is `True`. If set to `False`, the compiler does not validate the XPath queries.	NA

Table E–2 lists the configuration properties of sections of the `partnerLinkBinding` deployment descriptor. For each configuration property, a description is given as well as the expected behavior of Oracle BPEL Server when it is changed.

Table E–2 Configuration Properties for the partnerLinkBinding Deployment Descriptor

Property Name	Description	On Change
`callbackBindings`	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.	Recompile (not implemented)
`correlation`	Default value is `wsAddressing`. If this is set to `correlationSet`, this partner link is using the BPEL `correlationSet`.	If this is the process `partnerLink`, recompile (not implemented)
`contentType`	Sets special HTTP `contentType`. Example: `text/xml`	Takes effect immediately
`httpPassword`	For HTTP username/password authentication	Takes effect immediately
`httpUsername`	For HTTP username/password authentication	Takes effect immediately
`keepAlive`	If the server permits `keepAlive` connections, then this Boolean property can be turned on to take advantage of it. Thus, connections to the same server are shared between invocations.	Takes effect immediately
`location`	URL that overrides the location defined in the WSDL. For SOAP over HTTP binding, this value overrides the `SOAPAddress`.	Takes effect immediately
`nonBlockingInvoke`	Default value is `False`. When this is set to `True`, Oracle BPEL Server spawns a separate thread to do the invocation so that the invoke activity does not block the instance.	Takes effect immediately
`retryInterval`	Number of seconds that Oracle BPEL Server waits between retries.	Takes effect immediately
`retryMaxCount`	Number of retries that Oracle BPEL Server attempts, if an invoke fails because of network problems.	Takes effect immediately
`sendXSIType`	Some legacy RPC-style Web services require the `xsi:type` to be set with every element in the input message. If this value is set to `True`, Oracle BPEL Process Manager populates the `xsi:type` of all the elements.	Takes effect immediately

Table E–2 (Cont.) Configuration Properties for the partnerLinkBinding Deployment Descriptor

Property Name	Description	On Change
`serviceProperties`	--	Takes effect immediately
`timeout`	Number of seconds in which a SOAP call times out. A remote fault is thrown if this happens.	Takes effect immediately
`wsdlLocation`	URL of the WSDL file that defines this partner link. This property must be present. The BPEL compiler needs this to validate the BEPL source. This can be an abstract WSDL in that only the `portTypes` and their dependencies need to be defined in the WSDL.	Recompile (not implemented)
`wsdlRuntimeLocation`	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.	Clear WSDL cache (not implemented)

Summary

This appendix discusses deployment descriptor preference properties and deployment descriptor configuration properties, and how to set them in JDeveloper BPEL Designer.

F Demo User Community

This appendix describes the demo user community for task assignments in Oracle BPEL Process Manager.

This appendix contains the following topics:

- Setting Up JAZN Demo Users
- Summary

Setting Up JAZN Demo Users

Demo users are included with Oracle BPEL Process Manager. See "Demo Users and Roles" on page F-1 for a list of users and roles seeded with the product.

> **See Also:** "How-To: Configure Oracle's JAAS Provider with OC4J" at `http://www.oracle.com/technology/tech/java/oc4j/htdocs/how-to-security-JAAS.html`
>
> - Chapter 16, "Oracle BPEL Process Manager Workflow Services"
> - Chapter 17, "Worklist Application"

Demo Users and Roles

An Oracle BPEL Process Manager demo community is defined under the default realm `jazn.com`. It includes Oracle BPEL Process Manager system users `default`, `bpeladmin`, `guest` and seventeen other Oracle BPEL Process Manager users, which are shown in Table F–1. The default password is `welcome`.

Table F–1 Demo User Community

User	Common Name (cn)	Given Name	Middle Name	Surname (sn)	Title	Manager	E-mail
1	`achrist`	Agatha	--	Christie	Loan Consultant	`sfitzger`	`user3@dlsun1313.us.oracle.com`
2	`cdickens`	Charles	--	Dickens	CEO	--	`user1@dlsun1313.us.oracle.com`
3	`cdoyle`	Conan	--	Doyle	Loan Agent 2	`rsteven`	`user4@dlsun1313.us.oracle.com`
4	`fkafka`	Frants	--	Kafka	Manager 1	`ltolsoy`	`user2@dlsun1313.us.oracle.com`
5	`istone`	Irving	--	Stone	Loan Agent 2	`sfitzger`	`user3@dlsun1313.us.oracle.com`

Table F–1 (Cont.) Demo User Community

User	Common Name (cn)	Given Name	Middle Name	Surname (sn)	Title	Manager	E-mail
6	`jausten`	Jane	--	Austen	Loan Consultant	`fkafka`	`user3@dlsun1313.us.oracle.com`
7	`jcooper`	James	--	Cooper	Loan Agent 1	`jstein`	`user3@dlsun1313.us.oracle.com`
8	`jlondon`	Jack	--	London	Loan Agent 1	`sfitzger`	`user3@dlsun1313.us.oracle.com`
9	`jstein`	John	--	Steinbeck	Manager 2	`wfaulk`	`user2@dlsun1313.us.oracle.com`
10	`ltolsoy`	Leo	--	Tolstoy	Director	`wfaulk`	`user1@dlsun1313.us.oracle.com`
11	`mmitch`	Margaret	Munnerlyn	Mitchell	Loan Analyst	`fkafka`	`user3@dlsun1313.us.oracle.com`
12	`mtwain`	Mark	--	Twain	Loan Agent 2	`jstein`	`user3@dlsun1313.us.oracle.com`
13	`rsteven`	Robert	Louis	Stevenson	Manager 3	`jstein`	`user4@dlsun1313.us.oracle.com`
14	`sfitzger`	Scott	--	Fitzgerald	Manager 1	`wfaulk`	`user2@dlsun1313.us.oracle.com`
15	`szweig`	Stefan	--	Zweig	Loan Analyst	`fkafka`	`user3@dlsun1313.us.oracle.com`
16	`wfaulk`	William	--	Faulkner	Vice President	`cdickens`	`user1@dlsun1313.us.oracle.com`
17	`wshake`	William	--	Shakespeare	Loan Consultant	`rsteven`	`user4@dlsun1313.us.oracle.com`

All users have the `preferredLanguage` property defined as `en-US` (U.S. English) and the `orclWorkflowNotificationPref` property set to `Mail`.

Table F–2 and Table F–3 list the Oracle BPEL Process Manager roles and groups for the users shown in Table F–1.

Table F–2 Oracle BPEL Process Manager Roles

Roles	Direct Grantee Role	Users in Role
`BPMSystemAdmin`	--	`bpeladmin`
`BPMWorkflowReassign`	`BPMWorkflowAdmin`	`wfaulk, sfitzger, jstein, bpeladmin`
`BPMWorkflowSuspend`	`BPMWorkflowAdmin`	`wfaulk, sfitzger, jstein, bpeladmin`
`BPMWorkflowViewHistory`	`BPMWorkflowAdmin`	`wfaulk, bpeladmin`
`BPMWorkflowAdmin`	`BPMSystemAdmin`	`bpeladmin`
`BPMAnalyst`	`BPMWorkflowAdmin`	`sfitzger, jstein, guest, default, bpeladmin`
`DefaultBPMDomainAdmin`	`BPMSystemAdmin`	`default, bpeladmin`

Table F–3 Oracle BPEL Process Manager Groups

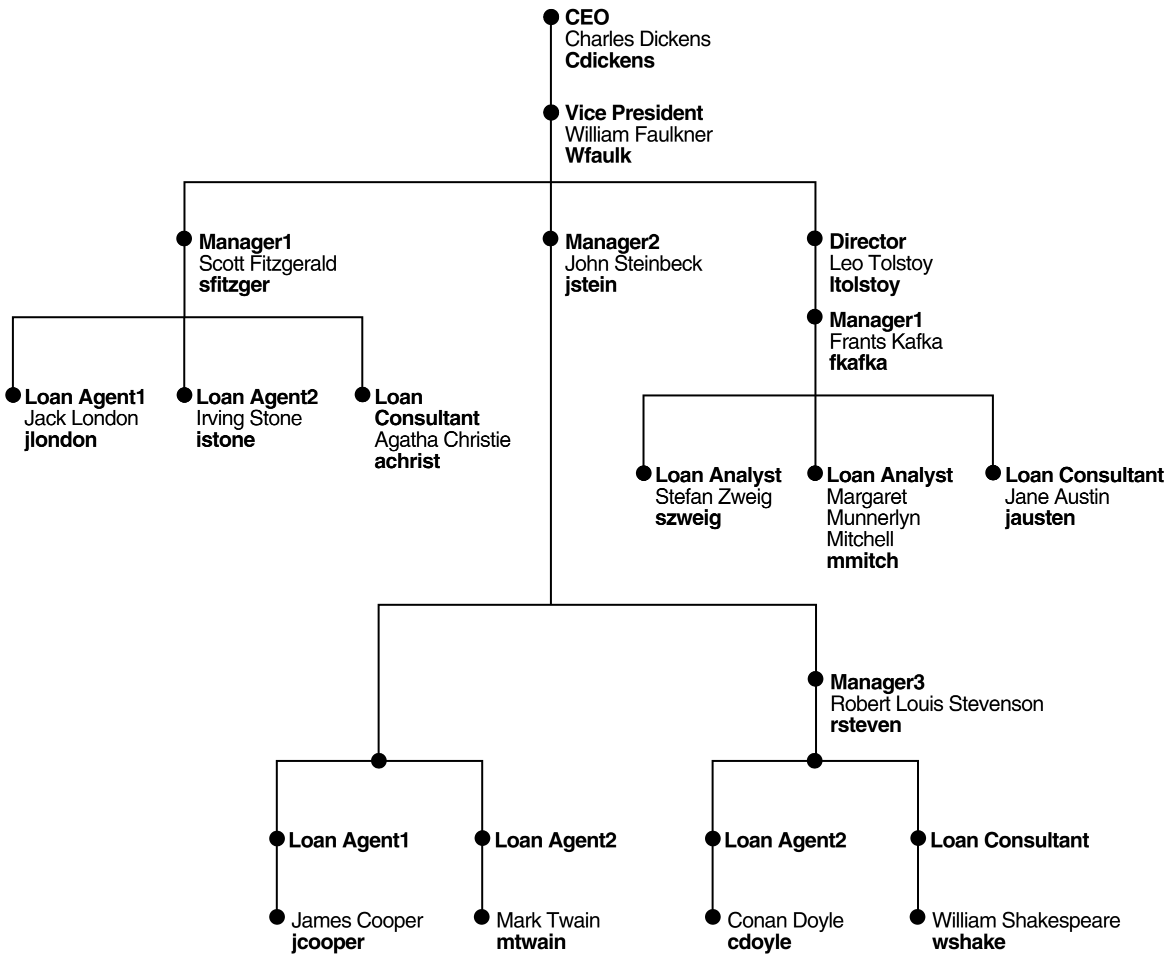
Groups	Direct Grantee Groups	Users in Group
`LoanAgentRole`	`Loan Analysis Group`	Directly: `jlondon`, `istone`, `jcooper`, `mtwain`, `cdoyle`, `wshake` Indirectly: `fkafka`, `szweig`, `mmitch`
`Loan Analysis Group`	--	Directly: `fkafka`, `szweig`, `mmitch`

As shown in Table F–3, the following users are not in any group or role: the CEO, `cdickens`, and the director, `ltolstoy`.

The Oracle BPEL Process Manager demo declares the `security` PUBLIC role. This role is implicitly granted to all registered Oracle BPEL Process Manager users.

Figure F–1 shows the organizational hierarchy of the demo users.

Figure F–1 Demo Users Organizational Hierarchy



Using the Demo User Community in the Order Booking Tutorial

The OrderBooking tutorial provides an example in which you use the demo user community. In this tutorial, you assign a group of users to a task with the Workflow wizard. When running this wizard, you perform the following tasks:

- Select **Sequential Workflow** as the workflow pattern.
- Select **Management Chain**, which enables a chain of management to sequentially review the task.
- Assign a task to a group named **Supervisor**.
- Select **1** for the number of levels in the management chain to sequentially review this task.

When the BPEL process is deployed, you log in to the Worklist Application with the user **jcooper**, acquire the task, and approve it. As shown in Table F–1, the supervisor of **jcooper** is **jstein**. Because you specified **1** as the number of levels in the management chain to sequentially review this task, **jstein** (the supervisor of **jcooper**), must also review this task. You then log in as **jstein** and approve the task.

See: *Oracle BPEL Process Manager Order Booking Tutorial* for instructions on using the Workflow wizard to perform these tasks

Summary

This appendix describes the demo user community for task assignments in Oracle BPEL Process Manager.

G

XPath Extension Functions

Oracle provides additional XPath functions that use built-in BPEL capabilities and XPath standards.

This appendix contains the following topics:

- XPath Extension Functions Available to BPEL Processes
- Summary

XPath Extension Functions Available to BPEL Processes

The following is an alphabetical list of additional XPath functions, along with the function descriptions, arguments, and other information.

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:

```
xp20: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`: `xp20`

add-dayTimeDuration-to-dateTime

This function returns a new datetime value adding `dateTime` to the given duration.

If 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-uri`: `http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.Xpath20`
- `namespace-prefix`: `xp20`

addChildNode

This function adds the World Wide Web consortium (W3C) document object model (DOM) child node to the incoming DOM element and returns the modified element.

Signature:

`ora:addChildNode(DOMElement element, Node node)`

Arguments:

- `DOMElement` - The DOM element to be modified
- `Node` - The DOM child note to add to the DOM Element

Property IDs

- `deprecated`

  Use the `bpelx:append` or `bpelx:insertBefore` extension activity to add or append a child node. This extension activity is demonstrated in the sample *`Oracle_Home`*`\integration\orabpel\samples\tutorials\126.DataAggregator`.
- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

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

- `namespace-uri`:`http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

appendToList

This function appends to a node list. The node list to be appended with 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 is created.

Property IDs

- `deprecated`

  Use the `bpelx:copyList` extension activity to append to a list. This extension activity is demonstrated in sample *`Oracle_Home`*`\integration\orabpel\samples\tutorials\126.DataAggregator`.
- `namespace-uri`: http://schemas.oracle.com/xpath/extension
- `namespace-prefix`: `ora`

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

Property IDs

- `namespace-uri`:`http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

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

Property IDs

- `namespace-uri`:`http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

clearTaskAssignees

This function clears the current task assignees.

Signature:

`ora:clearTaskAssignees(task)`

Arguments:

- `task` - The task

Property IDs

- `namespace-uri`:`http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

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`.

Example: `xp20:compare-ignore-case('Audi','bmw')` returns `-1`

Signature:

`orcl: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`: `http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.Xpath20`
- `namespace-prefix`: `xp20`

copyList

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

- `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)
- `partName` - The part to select from the variable (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 is created.

Property IDs

- `deprecated`
- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

countNodes

This function returns the number of the elements as an integer.

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`

create-delimited-string

This function returns a delimited string created from `nodeSet` delimited by delimiter.

Signature:

```
orcl:create-delimited-string(nodeSet as node-set, delimiter as
string)
```

Arguments:

- `nodeSet` - The node set to be converted into a deliminated string
- `delimiter` - The character that separates the items in the output string; for example, a comma or a semicolon.

Property IDs

- `namespace-uri`:
  `http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.servic`
  `es.functions.ExtFunc`
- `namespace-prefix`: `orcl`

create-nodeset-from-deliminated-string

The function takes a deliminated string and returns a `nodeSet`.

Signature:

```
orcl:create-nodeset-from-deliminated-string(qname,
deliminated-string, delimiter)
```

Arguments:

- `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`
- `delimited-string` - The sting of elements separated by the delimiter.
- `delimiter` - The character that separates the items in the input string; for example, a comma or a semicolon.

Property IDs

- `namespace-uri`: `http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.ExtFunc`
- `namespace-prefix`: `orcl`

createDeliminatedString

This function creates a delimited string from the arguments.

Signature:

```
ora:createDelimitedString('delimiter', Object)
```

Arguments:

- `delimiter` - the character that separates the items in the input string; for example, a comma or a semicolon.
- `Object` - The node set this function converts to a deliminated string.

Property IDs

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

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

- `namespace-uri`: `http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.Xpath20`
- `namespace-prefix`: `xp20`

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`

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

Property IDs

- `namespace-uri`: `http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.Xpath20`
- `namespace-prefix`: `xp20`

doc

This function returns content of an XML file.

Signature:

`ora:doc('fileName','xpath'?)`

Arguments:

- `fileName` - The name of the XML file
- `xpath` -

Property IDs

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

ends-with

This function returns true if inputString ends with searchString.

Example: `xp20:ends-with('XSL Map','Map')` returns `true`

Signature:

`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`: `http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.Xpath20`
- `namespace-prefix`: `xp20`

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`

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`

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-uri`: `http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.ExtFunc`
- `namespace-prefix`: `orcl`

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`

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 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`

generate-guid

Generates a unique GUID.

Signature:

```
orcl:generate-guid(object)
```

Arguments:

- `object` -

Property IDs

- `namespace-uri`: `http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.ExtFunc`
- `namespace-prefix`: `orcl`

generateGUID

Generates a unique GUID.

Signature:

`ora:generateGUID()`

Property IDs

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

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 as node-set` - The input element that the function returns as an XML representation

Property IDs

- `namespace-uri`: `http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.ExtFunc`
- `namespace-prefix`: `orcl`

get-localized-string

This function returns the locale-specific string for key.

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, key 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,resourceBundleName,language,country,variant,messageKey)`

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-uri`: `http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.ExtFunc`
- `namespace-prefix`: `orcl`

getChildElement

This function gets a child element for the given element

Signature:

`ora:getChildElement(element, index)`

Arguments:

- `element` - The source for the data
- `index` - Integer value of the child element index

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(element)`

Arguments:

- `element` - The source for the data

Property IDs

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

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` - Specifies a string formatted according to `java.text.SimpleDateFormat` format (optional).

Property IDs

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

getCurrentDateTime

This function returns the current date time as a string.

Signature:

`ora:getCurrentDateTime('format'?)`

Argument:

- `format` - Specifies a string formatted according to `java.text.SimpleDateFormat` format (optional).

Property IDs

- `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` - 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`

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`

getGroupProperty

This function retrieves the number of task attachments.

Signature:

`ora:getGroupProperty(groupId, attributeName)`

Arguments:

- `groupId` - String or element containing the group whose attribute should be retrieved
- `attributeName` - String or element containing the name of the group attribute. The attribute name should be one of the following values:
    1. `displayName`
    2. `mail`

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`

getLinkStatus

This function returns a Boolean 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-uri:`
  `http://schemas.xmlsoap.org/ws/2003/03/business-process/`
- `namespace-prefix: bpws`

getManager

This function gets the manager of a given user. If the user does not exist or if there is no manager for this user, it returns `null`.

Signature:

`ora:getManager(userID)`

Arguments:

- `userID` - The ID of the user for whom this function returns their manager

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`

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 get 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)
- `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`

getNumberOfTaskApprovals

This function computes the number of times the task was approved.

Signature:

`ora:getNumberOfTaskApprovals(taskId)`

Arguments:

- `taskId` - The ID of the task

Property IDs

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

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`

getPreviousTaskApprover

This function retrieves the previous task approver.

Signature:

`ora:getPreviousTaskApprover(taskId)`

Arguments:

- `taskId` - The ID of the task

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:

`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`

getReportees

This function gets the direct 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 called `user` and the namespace URI of the node is `http://oracle.tip.pc.services.identity/RemoteIdentityService.xsd`.

Signature:

`ora:getReportees(userId)`

Arguments:

- `userId` - The ID of the user

Property IDs

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

getTaskAttachmentByIndex

This function retrieves the task attachment at the specified index.

Signature:

`ora:getTaskAttachmentByIndex(taskId, attachmentIndex)`

Arguments:

- `taskId` - The task ID of the task
- `attachmentIndex` - The index of the attachment. The index begins from `1`.

Property IDs

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

getTaskAttachmentByName

This function retrieves the task attachment by the attachment name.

Signature:

`ora:getTaskAttachmentByName(taskId, attachmentName)`

Arguments:

- `taskId` - The task ID of the task.
- `attachmentName` - The name of the attachment.

Property IDs

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

getTaskAttachmentContents

This function retrieves the task attachment contents by the attachment name.

Signature:

`ora:getTaskAttachmentContents(taskId, attachmentName)`

Arguments:

- `taskId` - The task ID of the task.
- `attachmentName` - The name of the attachment.

Property IDs

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

getTaskAttachmentsCount

This function retrieves the number of task attachments.

Signature:

`ora:getTaskAttachmentsCount(taskId)`

Arguments:

- `taskId` - The task ID

Property IDs

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

getTaskAutoReleaseDuration

This function computes the release duration for the task when it is acquired.

Signature:

`ora:getTaskAutoReleaseDuration(taskId)`

Arguments:

- `taskId` - The ID of the task

Property IDs

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

getTaskReminderDuration

This function computes the next reminder to be sent for the task.

Signature:

`ora:getTaskReminderDuration(taskId)`

Arguments:

- `taskId` - The ID of the task

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`

getUserIdsFromGroupAlias

This function returns a `List` of user Ids for a group alias specified in the `TaskServiceAliases` section of the BPEL suitcase descriptor.

Signature:

`ora:getUserIdsFromGroupAlias( 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`

getUserProperty

This function retrieves the number of task attachments.

Signature:

`ora:getUserProperty(userId, attributeName)`

Arguments:

- `userId` - String or element containing the user whose attribute should be retrieved
- `attributeName` - String or element containing the name of the user attribute. The attribute name should be one of the following values:
  1. `givenName`
  2. `middleName`
  3. `sn`

4. `displayName`
5. `mail`
6. `telephoneNumber`
7. `homephone`
8. `mobile`
9. `facsimileTelephoneNumber`
10. `pager`
11. `preferredLanguage`
12. `manager`

Property IDs

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

getUserRoles

This function gets the user roles. This function returns a list of objects, either role object or group objects depending on the `roleType`.

The function returns a list of nodes. Each node in the list is called `group` or `role` depending on the `roleType` and the namespace URI of the node is `http://oracle.tip.pc.services.identity/RemoteIdentityService.xsd`.

Signature:

`ora:getUserRoles(userId, roleType, direct)`

Arguments:

- `userId` - String or element containing the user whose roles are to be retrieved
- `roleType` - The role type - should be one of the three values: `ApplicationRole`, `EnterpriseRole`, or `AnyRole`
- `direct` - String or element indicating if direct or indirect roles should be fetched. This is optional and if not specified only direct roles are fetched. This should be `xsd:boolean` or string `true/false`.

Property IDs

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

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` and the namespace URI of the node is `http://oracle.tip.pc.services.identity/RemoteIdentityService.xsd`.

Signature:

`ora:getUsersInGroup(groupId)`

Property IDs

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

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).

Property IDs

- `namespace-uri`:
  `http://schemas.xmlsoap.org/ws/2003/03/business-process/`
- `namespace-prefix`: bpws

getVariableProperty

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-uri`: `http://schemas.xmlsoap.org/ws/2003/03/business-process/`
- `namespace-prefix`: bpws

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.servic es.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.servic es.functions.Xpath20`
- `namespace-prefix`: `xp20`

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

- `namespace-uri`: `http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.ExtFunc`
- `namespace-prefix`: `orcl`

integer

This function returns content of node as integer.

Signature:

`ora:integer(node)`

Arguments:

- `node` - The input node

Property IDs

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

isUserInRole

This function verifies if a user has a given role, returning a Boolean `true` or `false`.

Signature:

`ora:isUserInRole(userId, roleName)`

Arguments:

- `userId` - String or element containing the user whose participation in the role should be verified
- `roleName` - The role name
- `direct` - String or element indicating if direct or indirect roles should be fetched. This is optional and if not specified only direct roles are fetched. This should be `xsd:boolean` or string `true/false`.

Property IDs

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

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:

`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`

lookupUser

This function returns LDAP user information.

Signature:

`ldap:lookupUser('properties','userId')`

Arguments:

- `properties` - The properties name as defined in the `directories.xml` file
- `userId` - The ID of the user for which this function returns LDAP information

Property IDs

- `namespace-uri`
  - **Value:** `http://schemas.oracle.com/xpath/extension/ldap`
- `namespace-prefix`
  - **Value:** `ldap`

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`

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`
  - **Value:** `http://schemas.oracle.com/xpath/extension/ldap`
- `namespace-prefix`
  - **Value:** `ldap`

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 datasource that can be either a JDBC connect string (`jdbc:oracle:thin:`*`username`*`/`*`password`*`@`*`host`*`:`*`port`*`:`*`sid`*) or a datasource JNDI identifier. Only Oracle Thin Driver is supported if the JDBC connect string is used.

Example: `orcl:lookup-table('employee','id','1234','last_ name','jdbc:oracle:thin:scott/tiger@localhost:1521:ORCL')`

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-uri:` `http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.servic es.functions.ExtFunc`
- `namespace-prefix`: `orcl`

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

- `namespace-uri`: `http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.ExtFunc`
- `namespace-prefix`: `orcl`

lookupUser

This function returns LDAP user information.

Signature:

```
ora:lookupUser(UserID)
```

Arguments:

- `object` - The string or node with the user name or ID

Property IDs

- `namespace-uri`
  - **Value:** `http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.ExtFunc`
- `namespace-prefix`
  - `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
  - `namespace-prefix`: `ora`

lookupGroup

This function looks up a group based on the arguments.

Signature:

`ora:lookupGroup(object)`

Arguments:

- `object` - The string containing the group name or ID

Property IDs

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

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`

matches

This function returns `true` if `intputString` matches the regular expression pattern `regexPattern`.

Example: `xp20:matches('abracadabra', '^a.*a$')` returns `true`

Signature:

`xp20:matches(intputString, 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`

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:

`orcl:max-value-among-nodeset(inputNumber as node-set)`

Arguments:

- `inputNumber` - The node-set of input numbers

Property IDs

- `namespace-uri`: `http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.ExtFunc`
- `namespace-prefix`: `orcl`

mergeChildNodes

Merges the child nodes of the incoming elements and returns the merged element.

Signature:

`ora:mergeChildNodes(DOMElement element1,DOMElement element2)`

Arguments:

- `variableName` - The source variable for the data
- `propertyName` - The QName of the property

Property IDs

- `deprecated`

  Use the `bpelx` extension activities to merge nodes. The extension activities are demonstrated in the sample.

  `Oracle_Home``\integration\orabpel\samples\tutorials\126.DataAggregator`.
- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

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.

Signature:

`orcl:min-value-among-nodeset(inputNumbers as node-set)`

Arguments:

- `inputNumber` - The node-set of input numbers

Property IDs

- namespace-uri: http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.ExtFunc
- namespace-prefix: orcl

minutes-from-dateTime

This function returns the minute from dateTime. The default minute is 0.

Signature:

```
xp20:minutes-from-dateTime(dateTime)
```

Arguments:

- dateTime - The dateTime

Property IDs

- namespace-uri: http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.Xpath20
- 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)
```

Arguments:

- dateTime - The dateTime to be formatted

Property IDs

- namespace-uri: http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.Xpath20
- namespace-prefix: xp20

parseEscapedXML

This function parses string to DOM.

Signature:

```
ora:parseEscapedXML(contentString)
```

Arguments:

- contentString - The string that this function parses to a DOM.

Property IDs

- namespace-uri: http://schemas.oracle.com/xpath/extension
- namespace-prefix: ora

processXSLT

This function returns the result of XSLT transformation.

Signature:

`ora:processXSLT('template','input','properties'?)`

Arguments:

- `template` - 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 results of the XSLT transformation by using the Oracle XDK XSLT processor.

Signature:

`xdk:processXSLT('template','input','properties'?)`

Arguments:

- `template` - XSLT template
- `input` - The input data to be transformed
- `properties` - The properties as defined in the `bpel.xml` file

Property IDs

- `namespace-uri`
  - **Value:** `http://schemas.oracle.com/bpel/extension/xpath/function/xdk`
- `namespace-prefix`
  - **Value:** `xdk`

processXSQL

This function returns the result of the XSQL request.

Signature:

`ora:processXSQL('template','input','properties'?)`

Arguments:

- `template` - 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`

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 a `<rowset>` element
- `row` - Indicates if each row should be enclosed in a `<row>` 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-uri`: `http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.ExtFunc`
- `namespace-prefix`: `orcl`

readFile

This function returns the content of the file.

Signature:

```
ora:readFile('fileName','nxsdTemplate'?,'nxsdRoot'?)
```

Arguments:

- `fileName` - The name of the file
- `nxsdTemplate` - The NXSD template for the output
- `nxsdRoot` -The NXSD root

Property IDs

- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

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`

search

This function returns a list of LDAP entries.

Signature:

`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`
  - **Value:** `http://schemas.oracle.com/xpath/extension/ldap`
- `namespace-prefix`
  - **Value:** `ldap`

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` - The `dateTime` as a string

Property IDs

- `namespace-uri`: `http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.Xpath20`
- `namespace-prefix`: xp20

sequence-next-val

Return the next value of an Oracle sequence.

The next value is obtained by executing

```
SELECT sequence.nextval FROM dual
```

against datasource that can be either a JDBC connect string (`jdbc:oracle:thin:`*`username`*`/`*`password`*`@`*`host`*`:`*`port`*`:`*`sid`*) or a datasource JNDI identifier. Only Oracle Thin Driver is supported if a JDBC connect string is used.

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` -
- datasource -

Property IDs

- `namespace-uri`: `http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.ExtFunc`
- `namespace-prefix`: `orcl`

setNodeValue

This function sets the variable's node value.

Signature:

```
ora:setNodeValue('variableName', 'part', 'query',
'newNodeValue')
```

Arguments:

- `variableName` - The source variable for the data
- `part` - The name of the part or message type
- `query` - The XPath expression used to search out the correct node
- `newNodeValue` - The new value of the node

Property IDs

- deprecated

  Use the `BPELX` extension activities to manipulate an XML document, this extension activities are demonstrated in sample

  `Oracle_Home``\integration\orabpel\samples\tutorials\126.DataAggregator`.
- `namespace-uri`: `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`: `ora`

square-root

This function returns the square root of `inputNumber`.

Example: `orcl:square-root(25)` returns `5`

Signature:

`orcl:square-root(inputNumber as number)`

Arguments:

- `inputNumber` - The input number for which the function calculates the square root

Property IDs

- `namespace-uri`: `http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.ExtFunc`
- `namespace-prefix`: `orcl`

subtract-dayTimeDuration-from-dateTime

This function returns a new `dateTime` value after subtracting duration from `dateTime`.

If 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 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 time zone

Property IDs

- `namespace-uri`: `http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.Xpath20`
- `namespace-prefix`: `xp20`

translateFromNative

Translates the input stream to an XML file.

Signature:

`ora:translateFromNative('string','nxsdTemplate'?,'nxsdRoot'?)`

Arguments:

- `string` - Data to be converted into an XML file.
- `nxsdTemplate` - The XSD file used to define how the translation is performed.
- `nxsdRoot` - 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` - XML file to be converted into a string.
- `nxsdTemplate` - The XSD file used to define how the translation is performed.
- `nxsdRoot` -Root element defined in the XSD file.

Property IDs

- `namespace-uri`
  - **Value:** `http://schemas.oracle.com/xpath/extension`
- `namespace-prefix`
  - **Value:** `ora`

upper-case

This function returns the value of inputString after translating every character to its upper-case correspondent.

Example: `xp20:upper-case('abCd0')` returns `'ABCD0'`

Signature:

`xp20:upper-case(inputString as string)`

Arguments:

- `inputString` - The input string

Property IDs

- `namespace-uri`: `http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.servic es.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`

Summary

This appendix lists the XPath extension functions; along with their descriptions, signature, argument descriptions, and property ID information.

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, 1-10
108.Timeouts
  tutorials, 1-10
109.CorrelationSets
  tutorials, 1-11
110.UserTasks
  tutorials, 1-11
111.CallingSessionBeans
  tutorials, 1-11
112.Arrays
  tutorials, 1-11
113.ABCARouting
  tutorials, 1-11
114.XSLTTransformations
  tutorials, 1-11
116.SendEmails
  tutorials, 1-11
117.ReceiveEmails
  tutorials, 1-11
118.JMSService
  tutorials, 1-11
119.JMSTopics
  tutorials, 1-11
120.XSQLExecution
  tutorials, 1-11
121.FileAdapter
  tutorials, 1-11
122.DBAdapter
  tutorials, 1-11
123.AQAdapter
  tutorials, 1-11
124.JMSAdapter
  tutorials, 1-11
125.ReportsSchema
  tutorials, 1-11
126.DataAggregator
  tutorials, 1-11
127.OrderBookingTutorial
  tutorials, 1-11
128.GoogleFlow
  tutorials, 1-11
129.FTPAdapter
  tutorials, 1-11
130.SendEmailWithAttachments
  tutorials, 1-11
701.LargeProcesses
  tutorials, 1-12
702.Bindings
  tutorials, 1-12

A

abs XPath extension function
  description, G-1
accountName
  XML file, 15-14
acquire
  task action, 16-30
acquired
  tasks, 16-21
acquiredByFunction
  approver function, 16-96
action permissions
  in worklists, 17-32
actionable notification
  definition, 15-1
actionableEmailAccount element
  configuring for e-mail approvals, 16-62
actions
  performed on a task, 16-29
activation agent
  e-mail, 15-14
activationAgent attributes
  setting for notifications in the bpel.xml file, 15-14
active
  process life cycle, 19-14
activities

adapters tab, C-3
correlation sets tab, C-3
definition, 1-3, 2-19
overview, 2-19, C-2
sensors tab, C-3
tasks common to many activities, C-3
Activities tab
Oracle BPEL Console, 19-21
activity interactions
viewing, 19-21
activity sensors
definition, 18-1
Adapter Configuration wizard
starting, 2-23
adapters
configuring, 2-23
definition, 2-23
documentation, 2-25
in JDeveloper BPEL Designer, 2-23
service names, 2-24
adapters tab
in activities, C-3
addChildNode XPath extension function
description, G-2
add-dayTimeDuration-to-dateTime XPath extension function
description, G-1
adding attachments
in worklists, 17-20
adding comments
in worklists, 17-19
addQuotes XPath extension function
description, G-2
adhoc workflow
changing the initial assignee of the task, 16-55
customizing, 16-55
definition, 16-9
implementation, 16-54
use case, 16-55
workflow pattern, 16-9
adhocFunction
approver function, 16-96
advanced searches
in worklists, 17-31
AmazonFlow
demos, 1-7
Ant
definition, 19-23
APIs
building a worklist application, 17-40
building a worklist application with the remote APIs, 17-45
appendToList XPath extension function
description, G-3
Applications Navigator
contents of, 2-5
definition, 2-5
location of in JDeveloper BPEL Designer, 2-4
approver functions, 16-95, 16-96
acquiredByFunction, 16-96
adhocFunction, 16-96
approverFunction, 16-95
groupsFunction, 16-96
listFunction, 16-96
managementChainFunction, 16-96
usersFunction, 16-96
approverFunction
approver function, 16-95
AQ adapter
tutorials, 1-11
arrays
appending new items to, 4-15
determining the size of, 4-12
in transformations, 14-13
manipulating, 4-10
maxOccurs attribute, 4-10
merging with the mergeChildNodes function, 4-15
statically indexing into, 4-11
tutorials, 1-11
assign activity
adding to an asynchronous service, 6-13
capabilities, C-3
copying data, 4-5
description, 3-11, 4-2
for data manipulation, 4-2
in asynchronous services, 6-13
in BPEL files, 3-11
inserting, 3-11
manually editing the workflow design, 16-11
references, 1-9
workflow task attributes defined within, 16-11
asynchronous callbacks, 6-3
asynchronous interaction with a notification time
BPEL process as the client, 13-6
BPEL process as the service, 13-6
asynchronous interaction with notification timer
definition, 13-5
asynchronous interaction with timeout
BPEL process as the client, 13-5
BPEL process as the service, 13-5
definition, 13-4
asynchronous interactions
BPEL process as the client, 13-4
BPEL process as the service, 13-4
definition, 13-3
returning faults, 9-8
asynchronous processes
retrieving status and results from, 12-5
using dehydration, 6-9
asynchronous services
assign activities, 6-13
callback responses with receive activities, 6-2
calling, 6-5, 6-9
correlating messages, 6-6
correlation IDs, 6-6
demos, 1-8
invoke activities, 6-5, 6-10
parallel flows, 7-2
partner links, 6-4, 6-5, 6-9

partnerLinkTypes, 6-4
receive activities, 6-5, 6-12
reply activities, 6-8
tutorials, 1-11
use case, 6-3
WS-Addressing, 6-6
attachments
sending, 15-6
tutorials, 1-11
attributes
manipulating, 4-9
audit trail
viewing, 3-19, 19-21
authenticate XPath extension function
description, G-3
authentication
through identity service, 16-75
authorization
through identity service, 16-75
auto mapping
in transformations, 14-14
auto mapping with confirmation
in transformations, 14-16

B

BankTransferDemo
demos, 1-7
batching
tutorials, 1-11
batchProcessActive XPath extension function
description, G-3
batchProcessCompleted XPath extension function
description, G-4
Boolean values
assigning, 4-8
BPEL
definition, 1-2
everything in XML format, 4-1
BPEL designers
definition, 2-2
overview of environments, 2-3
supported types of designers, 1-3
BPEL domains
creating and managing, 19-8
BPEL exec extension
references, 1-9
tutorials, 1-11
BPEL files
assign activities, 3-11
definition, 2-5
global variables, 3-10
in Hello World tutorial, 3-4
partner links, 3-9
partner links definition, 5-3
process element, 3-9
receive activities, 3-10
reply activities, 3-10
reviewing the contents of, 3-9
sequence activities, 3-11
sequences, 3-10
BPEL Inspector
contents of, 2-17
definition, 2-17
location of in Eclipse BPEL Designer, 2-15
BPEL Palette
contents of, 2-18
definition, 2-18
location of in Eclipse BPEL Designer, 2-15
BPEL processes
common interaction patterns, 13-2
compiling and deploying, 19-2
default revision, 19-15
invoking with a Web Service/SOAP interface, 12-6
life cycles, 19-14
sending messages to a process from a Java/JSP application, 12-2
undeployed, 19-15
viewing, 19-12
viewing and managing on the Oracle BPEL Console, 19-12
BPEL Processes tab
Oracle BPEL Console, 19-13
BPEL Source
location of in Eclipse BPEL Designer, 2-15
BPEL source
reviewing BPEL code, 3-9
BPEL Source window
contents of, 2-17
definition, 2-17
BPEL XPath functions
examples, 4-3
BPEL_ACTIVITY_SENSOR_VALUES
sensor public view, 18-5
BPEL_ALL_PROCESSES
sensor public view, 18-3
BPEL_ERRORS
sensor public view, 18-8
BPEL_FAULT_SENSOR_VALUES
sensor public view, 18-6
BPEL_PROCESS_ANALYSIS_REPORT
sensor public view, 18-3
BPEL_SENSOR_PROCESS_INSTANCES
sensor public view, 18-4
BPEL_VARIABLE_SENSOR_VALUES
sensor public view, 18-7
bpeladmin
user account, 16-75
bpelc
definition, 19-23
examples, 19-24
using, 19-23
bpelx exec built-in methods, 10-4
bpelx exec extension
built-in methods, 10-4
for embedding Java code in a BPEL process, 10-3
bpel.xml file
definition, 2-5
deployment descriptor file, E-2

in Hello World tutorial, 3-4
registering sensors and sensor actions, 18-19
setting activationAgent attributes, 15-14
BPMSystemAdmin
identity service role, 16-75
BPMWorkflowAdmin
identity service role, 16-75
BPMWorkflowReassign
identity service role, 16-75
BPMWorkflowSuspend
identity service role, 16-75
BPMWorkflowViewHistory
identity service role, 16-75
build.xml file
definition, 2-15
deploys the suitcase JAR file to the Oracle BPEL Server, 3-17
in Hello World tutorial, 3-4
business faults
definition, 9-4
business process execution language
See BPEL, 1-2

C

catch activity
adding, 9-12
capabilities, C-4
fault handling, 9-8
references, 1-9
channels
e-mail, 15-4
for task notification, 16-61
SMS, 15-12
voice mail, 15-10
CheckoutDemo
demos, 1-7
clearTaskAssignees extension function
description, G-4
compare extension function
description, G-4
compare-ignore-case extension function
description, G-5
compensate activity
capabilities, C-5
definition, 9-9
fault handling, 9-9
compilation
of BPEL processes, 19-2
without deployment on JDeveloper BPEL Designer, 19-4
compiling
in Hello World tutorial, 3-16
complex data transformations
recommendations on, 3-15
complex type
variables, 4-6
Component Palette
definition, 2-9
location of in JDeveloper BPEL Designer, 2-4
transformation functions, 2-9
concat
XPath function, 4-8
conditional branching logic
definition, 8-2
tutorials, 1-11
use case, 8-2
use of XPath expressions, 8-2
using switch activities, 8-2
using while activities, 8-4
conditional processing
with xsl choose, 14-12
with xsl if, 14-11
configuration properties
deployment descriptor, E-4
constant values
in transformations, 14-6
copyList extension function
description, G-5
core XPath functions
examples, 4-3
correlation ID
in asynchronous services, 6-2
WS-Addressing, 6-6
correlation sets
definition, 6-7, 6-8
demos, 1-8
tutorials, 1-11
correlation sets tab
in activities, C-3
correlations, 6-6
countNodes
XPath function, 4-12
countNodes extension function
description, G-6
create instance
definition, 6-13
in receive activities, 6-13
createDeliminatedString extension function
description, G-7
create-delimited-string extension function
description, G-6
create-nodeset-from-deliminated-string extension function
description, G-7
current-date extension function
description, G-7
current-dateTime extension function
description, G-8
current-time extension function
description, G-8
custom
sensor publish type, 18-2
custom plug-ins
configuration with identity service, 16-89
use with identity service, 16-76, 16-77

D

Dashboard tab

Oracle BPEL Console, 19-12
data manipulation
accessing fields with complex type variables, 4-6
appending new items to an array, 4-15
assigning Boolean values, 4-8
assigning date or time, 4-9
assigning literal strings, 4-7
assigning numeric values, 4-6
concatenating strings, 4-8
converting from a string to a structured XML object type, 4-16
copying data between variables, 4-5
determining array sizes, 4-12
dynamically indexing into a data sequence, 4-12
dynamically indexing with the getElement function, 4-14
initializing variables, 4-4
manipulating arrays, 4-10
manipulating attributes, 4-9
mathematical calculations with XPath functions, 4-7
merging arrays, 4-15
statically indexing into a data sequence, 4-11
with assign activities, 4-2
with the assign activity, 4-5
with XQuery and XSLT, 4-4
data sequences
dynamically indexing into, 4-12
database
sensor publish type, 18-2
database adapter
tutorials, 1-11
dates
assigning, 4-9
day-from-dateTime extension function
description, G-8
debatching
tutorials, 1-11
default
user account, 16-75
versions of processes, 19-15
default revision
of BPEL processes, 19-15
defining a fault handler, 9-11
dehydration
definition, 1-2, 6-9
delegations
of tasks, 16-7
deleting attachments
in worklists, 17-21
demo community
definition, F-1
for Oracle BPEL Process Manager, F-1
demonstrations
location of, 1-7
demos
AmazonFlow, 1-7
asynchronous services, 1-8
BankTransferDemo, 1-7
CheckoutDemo, 1-7
correlation sets, 1-8
DocumentReview, 1-8
FlowN activity, 1-8
GoogleFlow, 1-8
HelpDeskServiceRequest, 1-8
HotwireDemo, 1-8
IBMSamples, 1-8
Java embedding, 1-8
LoanDemo, 1-8
LoanDemoPlus, 1-8
LoanDemoPlusWithWorkflow, 1-8
OrderApproval, 1-8
parallel synchronous invocations, 1-8
ParallelSearch, 1-8
PaymentProcessor, 1-8
pick activity, 1-8
PriorityDemo, 1-8
ResilientDemo, 1-8
run time exceptions, 1-8
SalesforceFlow, 1-8
sensor actions, 18-9
sensors, 18-9
SleepBroker, 1-8
synchronous services, 1-8
TimeOffRequestDemo, 1-8
transformations, 1-9
VacationRequest, 1-9
while loop, 1-8
workflows, 1-8, 1-9
worklists, 1-8
XSLMapper, 1-9
XSLT Mapper, 1-9
deployment
deploy one process at a time, 19-2
in Hello World tutorial, 3-16
of a suitcase JAR file to a specific domain, 19-10
of BPEL processes, 19-2
of multiple versions of the same process, 19-3
overwriting versions, 19-3
status displays in Log Window, 19-3
to default domain, 19-2
undeploying a process from a specific domain, 19-11
with Eclipse BPEL Designer, 19-5
with JDeveloper BPEL Designer, 19-2
with the Invoke Deployment Tool, 19-3
deployment descriptor
bpel.xml file, E-2
configuration properties, E-4, E-5
defining a configuration property, E-4
defining a preference property, E-1
encrypting a preference value, E-3
getting a preference value within a BPEL process, E-3
preference properties, E-1
updating a preference at run time, E-2
use case, E-4
developer prompt
starting, 2-3
development mode

changing, 19-9
Diagram View window
definition, 2-6
location of in JDeveloper BPEL Designer, 2-4
dictionaries
in transformations, 14-18
display of changes
in worklists, 17-33
doc extension function
description, G-9
DocumentReview
demos, 1-8
domain passwords
changing, 12-3, 19-8
domains
best practices, 19-8
creating, 19-9
creating and managing, 19-8
definition, 19-3
deploying to specific domains, 19-10
location of a suitcase JAR file in a specific domain, 19-11
undeploying a process, 19-11
domain.xml file
increasing the syncMaxWaitTime property, 11-7
location of for Eclipse BPEL Designer, 11-7
location of for JDeveloper BPEL Designer, 11-7
dynamic assignments, 16-20
tutorial, 16-21
dynamic delimited strings
setting task assignees, 16-22
dynamic partner links
references, 1-9
dynamic workflow
See adhoc workflow

E

Eclipse BPEL Designer
BPEL Inspector, 2-17
BPEL Palette, 2-18
BPEL Source window, 2-17
compiling and deploying a process, 19-5
creating a project, 2-13
creating a workspaces, 2-13
defining a fault handler, 9-11
definition, 1-4
designing processes, 2-2
installation directory convention, 1-5
location of BPEL Inspector, 2-15
location of BPEL Palette, 2-15
location of BPEL Source, 2-15
location of domain.xml file, 11-7
location of Log Window, 2-15
location of Navigator, 2-15
location of Process Map, 2-15
Log window, 2-17
Navigator, 2-15
overview of design environment, 2-13
Overview window, 2-15
Process Map, 2-15
running the Hello World tutorial, 3-1
software download location, 1-5, 2-3, 2-15, 3-1
starting, 2-3
EJB
invoked by a BPEL process through use of WSIF binding, 10-2
tutorials, 1-12
e-mail
dynamically setting address, 15-13
notifications support, 15-2, 15-4
task action configuration, 16-74
tutorials, 1-11
e-mail activation agent
starting processes, 15-14
e-mail approval
for task notifications, 16-62
setting up the wf_config.xml file, 16-62
e-mail attachments
notifications support, 15-6
tutorial, 1-11
e-mail server
configuring for notifications, 15-8
configuring for notifications support, 15-8
e-mails
tutorials, 1-11
empty activity
capabilities, C-6
definition, 9-9
fault handling, 9-9
ends-with extension function
description, G-9
errors
in worklists, 17-28
invalid settings, C-2
escalate
task action, 16-30
escalations
of tasks, 16-7
evaluation time
definition, 18-10
event handlers
references, 1-9
exceptions, 9-4
tutorials, 1-11
expirations
of tasks, 16-7
expression constants
variable initialization, 4-4

F

facades
See XML facades, 10-3
fault handling, 9-11
adding a catch activity, 9-12
defining, 9-2, 9-11
defining a handler with Eclipse BPEL Designer, 9-11
definition, 9-2

importing RuntimeFault.wsdl, 9-11
modifying the WSDL files, 9-11
returning external faults, 9-7, 9-8
throwing internal faults, 9-7
tutorials, 1-10, 1-11
use case, 9-2
using catch activities, 9-8
using compensate activities, 9-9
using empty activities, 9-9
using scope activities, 9-4, 9-8
using terminate activities, 9-10
using throw activities, 9-7
fault sensors
definition, 18-2
faults
categories of faults in BPEL, 9-4
Qname fault name, 9-4
returning external faults, 9-7, 9-8
throwing internal faults, 9-7
file adapter
tutorials, 1-11
filters
sensor actions, 18-14
fire and forget
one-way message, 13-2
flex field updates
in worklists, 17-24
flex fields
code example, 16-15
in tasks, 16-15
flow activity
capabilities, C-6
defining, 7-3
references, 1-9
tutorials, 1-10
flowN activity
capabilities, C-7
definition, 7-4
demos, 1-8
references, 1-9
format extension function
description, G-9
formatDate extension function
description, G-10
format-dateTime extension function
description, G-10
format-string extension function
description, G-10
FTP adapter
tutorials, 1-11
function
descriptions, 14-7
functions
approver function examples for workflows, 16-96
approver functions for workflows, 16-95
chaining in transformations, 14-9
editing in transformations, 14-8
editing XPath expressions in transformations, 14-9
in transformations, 14-7
location of XPath extension function descriptions, 4-3
user-defined in transformations, 14-9
XPath extension functions prefixed with xp20 or orcl, 14-7
FYI task
cannot be extended, 16-57
changing the assigning after creating the task, 16-56
customizing, 16-56
definition, 16-9
implementation, 16-55
use case, 16-55
workflow pattern, 16-9

G

genEmptyElem extension function
description, G-11
generateGUID extension function
description, G-11
generate-guid extension function
description, G-11
getChildElement extension function
description, G-13
getContentAsString extension function
description, G-13
get-content-as-string extension function
description, G-12
getConversationId extension function
description, G-13
getCreator extension function
description, G-14
getCurrentDate, 4-9
getCurrentDate extension function
description, G-14
getCurrentDateTime, 4-9
getCurrentDateTime extension function
description, G-14
getCurrentTime, 4-9
getCurrentTime extension function
description, G-14
getDomainId extension function
description, G-15
getElement
XPath function, 4-14
getElement extension function
description, G-15
getGroupIdsFromGroupAlias extension function
description, G-15
getGroupProperty extension function
description, G-16
getInstanceId extension function
description, G-16
getLinkStatus extension function
description, G-16
get-localized-string extension function
description, G-12
getManager extension function
description, G-17

getMessage extension function
  description, G-17
getNodes extension function
  description, G-18
getNodeValue extension function
  description, G-17
getNumberOfTaskApprovals extension function
  description, G-18
getPreference extension function
  description, G-18
getPreviousTaskApprover extension function
  description, G-18
getProcessId extension function
  description, G-19
getProcessOwnerId extension function
  description, G-19
getProcessURL extension function
  description, G-19
getProcessVersion extension function
  description, G-19
getReportees extension function
  description, G-20
getTaskAttachmentByIndex extension function
  description, G-20
getTaskAttachmentByName extension function
  description, G-20
getTaskAttachmentContents extension function
  description, G-21
getTaskAttachmentsCount extension function
  description, G-21
getTaskAutoReleaseDuration extension function
  description, G-21
getTaskReminderDuration extension function
  description, G-21
getUserAliasId extension function
  description, G-22
getUserIdsFromGroupAlias extension function
  description, G-22
getUserProperty extension function
  description, G-22
getUserRoles extension function
  description, G-23
getUsersInGroup extension function
  description, G-23
getVariableData, 4-8
  assigning tasks dynamically with, 16-20
getVariableData extension function
  description, G-24
getVariableData function
  using in mathematical calculations, 4-7
getVariableProperty extension function
  description, G-24
global variables
  in BPEL files, 3-10
GoogleFlow
  demos, 1-8
grantees
  relationship with roles, 16-76
group information
  in worklists, 17-29
groups
  creating, 16-77
groupsFunction
  approver function, 16-96
guest
  user account, 16-75

H

Hello World tutorial
  adding an assign activity, 3-12
  browsing the new project, 3-4
  compiling and deploying the BPEL process, 3-16
  creating a new project, 3-2
  editing the WSDL file, 3-5
  overview, 3-1
  reviewing the BPEL source code, 3-9
  running, 3-1
  suitcase JAR file, 3-17
  testing, 3-17
  viewing the project files, 3-4
HelpDeskServiceRequest
  demos, 1-8
HotwireDemo
  demos, 1-8
hours-from-dateTime extension function
  description, G-25
HTTP get method
  tutorials, 1-12

I

iaswconfig.xml file
  location of, 15-11
IBMSamples
  demos, 1-8
IDeliveryService.post() method
  invoking a one-way Web service operation, 12-5
IDeliveryService.request() method
  invoking a two-way Web service operation, 12-4
identity service, 16-2
  configuring the is_config.xml file, 16-79
  connection element, 16-80
  connection pool properties, 16-81
  creating users and groups, 16-77
  custom plug-ins configuration, 16-89
  definition, 16-2, 16-75
  determining a user's local language and time zone, 17-33
  LDAP-based (Oracle Internet Directory) JAZN provider configuration, 16-83
  middle-tier configuration, 16-85
  providers, 16-77
  roles, 16-75
  structure of identity service configuration file, 16-79
  support for in workflow, 16-2
  third-party LDAP server configuration, 16-86
  use with custom plug-ins, 16-76, 16-77
  use with JAZN, 16-2, 16-75, 16-76

use with LDAP, 16-76
use with third-party LDAP servers, 16-77
user and role properties, 16-78
userControls and roleControl elements, 16-81
users, 16-75
XML-based JAZN provider configuration, 16-83
Identity service component
definition, 16-5, B-3
WSDL operations, B-3
identity services
providers, 16-76
implicit-timezone extension function
description, G-25
indexing methods
using XPath, 4-11
index-within-string extension function
description, G-25
Instances tab
Oracle BPEL Console, 19-21
integer extension function
description, G-26
interaction patterns
a valid partner link is required, 13-2
asynchronous interaction with notification timer, 13-5
asynchronous interaction with timeout, 13-4
asynchronous interactions, 13-3
common patterns between a BPEL process and another application, 13-2
one request, a mandatory response, and an optional response, 13-8
one request, multiple responses, 13-6
one request, one of two possible responses, 13-7
one-way message, 13-2
partial processing, 13-9
synchronous interactions, 13-2
third-interactions, 13-10
interop
location of, 1-7
invalid settings error, C-2
invoke, 6-5
invoke activity
adding to an asynchronous service, 6-10
capabilities, C-8
definition, 5-2
in asynchronous services, 6-2, 6-5, 6-10
in synchronous services, 5-2, 5-5
manually editing the workflow design, 16-11
references, 1-9
tutorials, 1-10
Invoke Deployment Tool
deploying, 19-3
is_config.xml file
location of, 16-79
isUserInRole extension function
description, G-26

J

Java API
invoking a BPEL process with the generic Java API, 12-3
invoking a one-way operation, 12-5
invoking a two-way operation, 12-4
NormalizedMessage class for dynamically activating messages, 12-3
parsing XML messages, 12-3
using from a remote client, 12-6
using the Locator class to connect to Oracle BPEL Process Manager, 12-3
Java applications
wrapped as SOAP services, 10-2
Java embedding
bpelx exec extension, 10-4
demos, 1-8
embedding code in a BPEL process, 10-3
example using JDeveloper BPEL Designer, 10-5
in a BPEL process, 10-2
invoking an EJB, 10-2
Java application requires a BPEL compatible interface, 10-2
tutorials, 10-2
use case, 10-2
using bpelx exec, 10-3
using WSIF binding, 10-2
Java embedding activity
capabilities, C-9
Java method
tutorials, 1-12
Javadocs
for JSP forms, 16-73
location of, 12-4, 16-73, 16-77
JavaExec
references, 1-9
Java/JSP applications
calling a BPEL process, 12-2
JAZN, 16-2
configuration of LDAP-based (Oracle Internet Directory) provider, 16-83
configuration of XML-based provider, 16-83
definition, 16-76
demo users, F-1
demo users and roles, F-1
LDAP-based (Oracle Internet Directory) provider type, 16-77
middle-tier configuration of LDAP-based JAZN provider, 16-85
setting up users, F-1
storing a user's local language and time zone, 17-33
used with identity service, 16-75, 16-76
used with third-party LDAP servers, 16-77
user notification preferences, 16-61
XML-based provider type, 16-77
jazn-data.xml file
creating users and groups with XML-based JAZN provider, 16-77
jazn.xml file
configuring, 16-85
JBoss

JMS, 1-11
JDeveloper BPEL Designer
adapters, 2-23
Applications Navigator, 2-5
compiling and deploying a process, 19-2
compiling without deploying, 19-4
Component Palette, 2-9
configuring notifications, 15-2
creating a project, 2-3
creating a workspaces, 2-3
creating sensors, 18-9
definition, 1-3
designing a simple workflow, 16-31
designing processes, 2-2
designing timeouts, 11-4
designing workflows, 16-10
Diagram View window, 2-6
location of Applications Navigator, 2-4
location of Component Palette, 2-4
location of Diagram View window, 2-4
location of domain.xml file, 11-7
location of Log window, 2-5
location of Process Activities, 2-4
location of Property Inspector, 2-4
location of Structure window, 2-5
Log window, 2-12
notifications, 2-23
overview of design environment, 2-3
overview of services, 2-22
Process Activities, 2-9
Property Inspector, 2-10
sensors, 2-23
Source window, 2-8
starting, 2-2
Structure window, 2-11
transformations, 2-23, 14-2
using Java embedding, 10-5
workflows, 2-23
worklists, 2-23
JMS
tutorials, 1-11
with JBoss, 1-11
with OC4J, 1-11
with WebLogic, 1-11
JMS adapter
tutorials, 1-11
JMSQueue
sensor publish type, 18-2
JMSTopic
sensor publish type, 18-2
JSP
tutorials, 1-10
JSP form
custom JSP display, 16-72, 16-73
custom Oracle BPEL Worklist Application display, 16-72, 16-73
customizing the default JSP display, 16-67
customizing the default JSP file, 16-69
customizing the default XML mapping file, 16-67
default file name and location, 16-65, 16-71
default mapping XML file name and location, 16-65
default Oracle BPEL Worklist Application display, 16-12, 16-65
deploying the default JSP to OC4J, 16-71
multibyte payload in the task detail JSP, 16-71

L

languages
accessing the Oracle BPEL Worklist Application in local languages, 17-33
setting in JAZN, 17-33
setting in LDAP, 17-33
last-index-within-string extension function
description, G-26
LDAP
configuration of third-party LDAP servers, 16-86
storing a user's local language and time zone, 17-33
used with identity service, 16-2, 16-76
left-trim extension function
description, G-27
life cycles
default revision, 19-15
of processes, 19-13, 19-14
process recommendations for a development environment, 19-15
process recommendations for a production environment, 19-15
undeployed, 19-15
use case, 19-16
links
references, 1-9
listFunction
approver function, 16-96
listUsers extension function
description, G-28
literal strings
assigning, 4-7
literal XML
variable initialization, 4-4
LoanDemo
demos, 1-8
LoanDemoPlus
demos, 1-8
LoanDemoPlusWithWorkflow
demos, 1-8
LocalBPELServer
default Oracle BPEL Server connection, 19-2
Log window
contents of, 2-17
definition, 2-12, 2-17
location of in Eclipse BPEL Designer, 2-15
location of in JDeveloper BPEL Designer, 2-5
lookupGroup extension function
description, G-29
lookup-table extension function
description, G-28
lookupUser extension function

description, G-27, G-29
lookup-xml extension function
description, G-29
lower-case extension function
description, G-30

M

managementChainFunction
approver function, 16-96
matches extension function
description, G-30
maxOccurs attribute, 4-10, 4-11
max-value-among-nodeset extension function
description, G-30
mergeChildNodes
XPath function, 4-15
mergeChildNodes extension function
description, G-31
message flow
tutorials, 1-11
minutes-from-dateTime extension function
description, G-32
min-value-among-nodeset extension function
description, G-31
modes
changing for the Oracle BPEL Server, 19-9
month-from-dateTime extension function
description, G-32
multibyte payload
in the task detail JSP, 16-71

N

Navigator
contents of, 2-15
definition, 2-15
location of in Eclipse BPEL Designer, 2-15
New Project wizard, 3-2
NormalizedMessage class
for dynamically activating messages, 12-3
specifying the address of a Web service for the callback, 12-6
NormalizedMessages class
viewing the specific field name for the conversation ID, 12-6
notification activity
capabilities, C-10
using, 15-2
notification message variables
for task notification, 16-62
notification messages
for task notification, 16-61
Notification service component
definition, 16-5, B-2
WSDL operations, B-2
notification services
definition, 16-3
notifications
configuring in JDeveloper BPEL Designer, 15-2
configuring wireless service provider support for SMS, 15-13
configuring wireless service provider support for voice, 15-11
definition, 2-23, 15-1, 16-2
dynamically setting e-mail addresses and telephone numbers, 15-13
e-mail attachment support, 15-6
e-mail attachment tutorials, 1-11
e-mail server configuration, 15-8
e-mail support, 15-2, 15-4
in JDeveloper BPEL Designer, 2-23
selecting recipients by browsing the user directory, 15-13
SMS support, 15-12
tutorials, 1-11
use case, 15-1
using Oracle Application Server Wireless, 15-2
voice mail support, 15-10
notifications and reminders
in tasks, 16-19, 16-60
ns_emails.xml file
example, 15-9
location of, 15-8
ns_iaswconfig.xml file
example, 15-11
location of, 15-13
numeric values
assigning, 4-6

O

obant
running, 3-17
OC4J
deploying the default JSP to, 16-71
JMS, 1-11
off
process state, 19-14
OID
See Oracle Internet Directory, 16-83
on
process state, 19-14
onAlarm branch
of pick activity, 11-3
one request, a mandatory response, and an optional response
BPEL process as the client, 13-9
BPEL process as the service, 13-9
definition, 13-8
one request, multiple responses
BPEL process as the client, 13-7
BPEL process as the service, 13-7
definition, 13-6
one request, one of two possible responses
BPEL process as the client, 13-8
BPEL process as the service, 13-8
definition, 13-7
one-way message
BPEL process as the client, 13-2

BPEL process as the service, 13-2
definition, 13-2
onMessage branch
of pick activity, 11-3
ora
getGroupProperty
XPath extension function, 16-91
getManager
XPath extension function, 16-91
getNumberOfTaskApprovals
XPath extension function, 16-92
getPreviousTaskApprover
XPath extension function, 16-92
getReportees
XPath extension function, 16-92
getTaskAttachmentByIndex
XPath extension function, 16-93
getTaskAttachmentByName
XPath extension function, 16-94
getTaskAttachmentContents
XPath extension function, 16-94
getTaskAttachmentsCount
XPath extension function, 16-93
getUserInRole
XPath extension function, 16-92
getUserProperty
XPath extension function, 16-91
getUserRoles
XPath extension function, 16-92
getUsersInGroup
XPath extension function, 16-92
lookupGroup
XPath extension function, 16-90
lookupUser
XPath extension function, 16-89
Oracle Application Server Wireless
wireless and voice component, 15-2
Oracle BPEL Admin Console
accessing, 19-8
changing passwords, 19-8
Oracle BPEL Console
accessing, 19-8
Activities tab, 19-21
BPEL Processes tab, 19-13
Dashboard tab, 19-12
Instances tab, 19-21
location of XPath extension function descriptions, 4-3
overview, 2-21
running and managing processes from, 2-2
starting, 2-3
testing the Hello World tutorial, 3-17
viewing a history of process activities, 19-21
viewing activity interactions, 19-21
viewing and managing processes, 19-12
viewing sensor and sensor action metadata, 18-19
viewing sensor values, 19-21
viewing the audit trail, 3-19
viewing the audit trail of a process, 19-21
viewing the visual flow, 3-19
Oracle BPEL Process Manager
adds value and ease of use to BPEL functionality, 1-3
components of, 2-2
definition, 1-2
provides support for, 1-2
services in JDeveloper BPEL Designer, 2-22
starting, 2-2
supported designer types, 1-3
Oracle BPEL Server
changing modes, 19-9
deploying process to, 2-2
LocalBPELServer default connection, 19-2
overview, 2-21
starting, 2-2
Oracle BPEL Worklist Application
accessing in local languages, 17-33
accessing tasks assigned to users, 16-8
adding attachments, 17-20
adding comments, 17-19
advanced searches, 17-31
architecture, 17-34
cannot view User Task 2.0 Macro tasks, 16-56
concepts, 17-2
custom JSP display form, 16-72, 16-73
customizing, 17-34
default JSP display form, 16-12, 16-65
definition, 16-3
deleting attachments, 17-21
determining action permissions, 17-32
errors, 17-28
features, 17-4
flex and task field updates, 17-24
home page contents, 17-6
logging into, 17-4
parallel tasks, 17-23
payload viewing, 17-17
payload viewing of source XML, 17-18
request status, 17-26
requesting more information, 17-13
responding to tasks from, 16-1
routing, 17-12
starting, 2-3
Task Actions list, 17-10
Task Details page contents, 17-9
task history, 17-14
task reassignment, 17-22
use cases, 17-1
user and group information, 17-29
visual display of changes, 17-33
XSLT payload display, 16-71
Oracle Delegated Administration Services
creating users and groups, 16-77
setting user properties for notification, 16-61
Oracle Internet Directory
configuration with identity service, 16-83
creating users and groups, 16-77
LDAP-based provider type, 16-77
object classes for specifying user and role properties, 16-78

storing a user's local language and time zone, 17-33
orcl
create-nodeset-from-delimited-string, 16-22
format-string
XPath extension function, 16-95
get-localized-string
XPath extension function, 16-95
get-localized-string()
XPath function, 16-19
OrderApproval
demos, 1-8
outcome-based modeling, 16-59
case statements for other task conclusions, 16-59
payload updates, 16-59
overview, 18-1
Overview window
contents of, 2-15
definition, 2-15

P

parallel flows
defining, 7-3
definition, 7-2
tutorials, 1-10, 1-11
use case, 7-2
parallel synchronous invocations
demos, 1-8
parallel tasks, 16-21
in worklists, 17-23
parallel workflow
changing the outcomes after creation of the use task, 16-52
changing the parameters of the outcome determination policy, 16-52
customizing, 16-52
definition, 16-9
implementation, 16-49
pattern-specific parameters, 16-51
use case, 16-51
workflow pattern, 16-9
parallel workflow with final reviewer
changing the final reviewer of the task, 16-53
customizing, 16-53
definition, 16-9
implementation, 16-52
pattern-specific parameters, 16-53
use case, 16-53
workflow pattern, 16-9
ParallelSearch
demos, 1-8
parseEscapedXML extension function
description, G-32
parseEscapeXML
XPath function, 4-16
partial processing
BPEL process as the client, 13-10
BPEL process as the service, 13-10
definition, 13-9
partner links
adding to an asynchronous service, 6-9
BPEL file code example, 5-3
definition, 1-3, 2-20, 5-3
displaying in Diagram View window, 2-8
dynamic, 1-9
in asynchronous services, 6-4, 6-5, 6-9
in BPEL files, 3-9
in synchronous services, 5-2
overview, 2-20
specifying a WSDL file, 2-21
UDDI and WSIL directories, 5-5
partnerLink activity
capabilities, C-10
partnerLinkTypes
definition, 5-4, 6-4
in asynchronous services, 6-4
in synchronous services, 5-4
passwords
changing domain passwords, 19-8
changing for the Oracle BPEL Admin Console, 19-8
patterns
of interaction between a BPEL process and another application, 13-2
payload
workflow task, 16-11
payload display
workflow task, 16-12, 16-64, 16-71
payload viewing
in worklists, 17-17
payload viewing of source XML
in worklists, 17-18
PaymentProcessor
demos, 1-8
pc.properties file
location of, 16-74
pick activity
capabilities, C-11
code example, 11-3
demos, 1-8
for continuing with a flow after a certain amount of time, 11-2
for timeouts, 11-2
onAlarm branch, 11-3
onMessage branch, 11-3
references, 1-9
tutorials, 1-10
two condition branches, 11-2
port types
definition, 5-4
in asynchronous services, 6-4
in synchronous services, 5-4
ports
in synchronous services, 5-2
position
XPath function, 4-11
post() method
support for client specifying a conversation ID, 12-6

preference properties
  deployment descriptor, E-1
PriorityDemo
  demos, 1-8
Process Activities
  definition, 2-9
  location of in JDeveloper BPEL Designer, 2-4
process element
  in BPEL files, 3-9
process history
  viewing, 19-21
Process Map
  contents of, 2-15
  definition, 2-15
  location of in Eclipse BPEL Designer, 2-15
processXSLT extension function
  description, G-33
processXSQL extension function
  description, G-33
procurement process
  workflow functionality example, 16-3
production mode
  changing, 19-9
.project file
  definition, 3-4
project files
  BPEL file, 2-5
  bpel.xml, 2-5
  in Applications Navigator, 2-5
  in Hello World tutorial, 3-4
  in Navigator, 2-15
  WSDL file, 2-5
projects
  creating for the Hello World tutorial, 3-2
  creating in Eclipse BPEL Designer, 2-13
  creating in JDeveloper BPEL Designer, 2-3
  definition, 2-3
  naming limitations, 2-3, 2-13
Property Inspector
  definition, 2-10
  location of in JDeveloper BPEL Designer, 2-4
PUBLIC
  identity service role, 16-75
public views
  BPEL_ACTIVITY_SENSOR_VALUES, 18-5
  BPEL_ALL_PROCESSES, 18-3
  BPEL_ERRORS, 18-8
  BPEL_FAULT_SENSOR_VALUES, 18-6
  BPEL_PROCESS_ANALYSIS_REPORT, 18-3
  BPEL_SENSOR_PROCESS_INSTANCES, 18-4
  BPEL_VARIABLE_SENSOR_VALUES, 18-7
  sensors, 18-2
publish types
  creating a custom publisher, 18-15
  custom, 18-2
  database, 18-2
  definition, 18-2
  JMSQueue, 18-2
  JMSTopic, 18-2

Q

Qname
  fault name, 9-4
query-database extension function
  description, G-34

R

readFile extension function
  description, G-34
reassign
  task action, 16-30
receive activity
  adding to an asynchronous service, 6-12
  capabilities, C-12
  create instance, 6-13
  in asynchronous services, 6-2, 6-5, 6-12
  in BPEL files, 3-10
  references, 1-9
  tutorials, 1-10
references
  assign activity, 1-9
  BPEL exec extension, 1-9
  catch activity, 1-9
  dynamic partner links, 1-9
  event handlers, 1-9
  flow activity, 1-9
  flowN activity, 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 activity, 1-9
  reply activity, 1-9
  switch activity, 1-9
  terminate activity, 1-10
  throw activity, 1-10
  wait activity, 1-10
  while activity, 1-10
  XPath function, 1-9
  XPath functions, 1-10
  XPath queries, 1-10
release
  task action, 16-30
reminders
  for task notifications, 16-63
renew
  task action, 16-30
repeating elements
  in transformations, 14-13
replay activity
  references, 1-9
reply activity
  capabilities, C-13
  definition, 6-8
  in asynchronous services, 6-8
  in BPEL files, 3-10
  references, 1-9

reporting schema
  for database publish type of sensors, 18-2
reports
  generating for transformations, 14-20
  tutorials, 1-11
request more information
  task action, 16-30
request more information with reapproval
  task action, 16-30
request status
  in worklists, 17-26
requesting more information
  in worklists, 17-13
ResilientDemo
  demos, 1-8
resource bundles
  in tasks, 16-19
restricted actions
  code example, 16-16
  in tasks, 16-16
resume
  task action, 16-30
retired
  process life cycle, 19-14
right-trim extension function
  description, G-34
RMI
  tutorials, 1-10
  using Java API through, 12-6
roleControls Element, 16-81
roles
  BPMSystemAdmin, 16-75
  BPMWorkflowAdmin, 16-75
  BPMWorkflowReassign, 16-75
  BPMWorkflowSuspend, 16-75
  BPMWorkflowViewHistory, 16-75
  for partner links in asynchronous services, 6-4
  predefined, 16-75
  predefined by identity service, 16-75
  properties, 16-78
  PUBLIC, 16-75
  relationship with grantees, 16-76
route
  task action, 16-30
routing
  in worklists, 17-12
RPC styles
  differences with document-literal styles in WSDL files, 4-1, 4-16
run-time exceptions, 9-4
  demos, 1-8
run-time faults
  definition, 9-4
RuntimeFault.wsdl file
  importing into a process, 9-11

S

samples
  location of, 1-7
schemac
  definition, 19-24
  examples, 19-25
  generating facades from WSDL or XSD files, 10-3
  generating XML facades from WSDL or XSD files, 19-24
  using, 19-24, D-14
scope activity
  capabilities, C-15
  definition, 9-4
  fault handling, 9-4, 9-8
search extension function
  description, G-35
seconds-from-dateTime extension function
  description, G-35
sensor actions
  configuring, 18-12
  creating with a filter, 18-14
  demos, 18-9
  registering in the bpel.xml file, 18-19
  viewing definitions, 18-19
  viewing metadata from the Oracle BPEL Console, 18-19
sensor data
  persisting in a reporting schema, 18-2
sensor values
  viewing, 19-21
sensorAction.xml file, 18-13, 18-14
sensors, 18-1
  activity sensors, 18-1
  BPEL reporting schema, 18-2
  BPEL_ACTIVITY_SENSOR_VALUES public views, 18-5
  BPEL_ALL_PROCESSES public views, 18-3
  BPEL_ERRORS public views, 18-8
  BPEL_FAULT_SENSOR_VALUES public views, 18-6
  BPEL_PROCESS_ANALYSIS_REPORT public views, 18-3
  BPEL_SENSOR_PROCESS_INSTANCES public views, 18-4
  BPEL_VARIABLE_SENSOR_VALUES public views, 18-7
  configuring, 18-10
  creating a custom publish type, 18-15
  creating in JDeveloper BPEL Designer, 18-9
  definition, 2-23, 18-1
  demos, 18-9
  evaluation time, 18-10
  fault sensors, 18-2
  in JDeveloper BPEL Designer, 2-23
  public views, 18-2
  publish types, 18-2
  registering in the bpel.xml file, 18-19
  sensorAction.xml file, 18-2
  sensor.xml file, 18-2
  tutorials, 1-11
  use cases, 18-1
  variable sensors, 18-1
  viewing definitions, 18-19

viewing metadata from the Oracle BPEL Console, 18-19
sensors tab
in activities, C-3
sensor.xml file, 18-11
sequence activity
capabilities, C-14
description, 3-11
in BPEL files, 3-11
sequence-next-val extension function
description, G-35
sequences
in BPEL files, 3-10
sequential tasks, 16-20
sequential workflow
adding approvers, 16-46
adding the continue routing expression, 16-46
approver function examples, 16-96
approver functions, 16-95
approvers assignment policy, 16-43
changing routing policy, 16-46
changing the assignees in the list after creating the task, 16-46
changing the continue routing expression, 16-46
changing the initial assignee after creating the task, 16-45
changing the management chain parameters after task creation, 16-45
changing the outcome after task creation, 16-46
changing the user or group to whom the task is routed based on output of an external system, 16-46
continue routing expression, 16-43
customizing, 16-45
definition, 16-9
implementation, 16-40
outcome that results in the task being routed, 16-43
pattern-specific parameters, 16-43
use cases, 16-43
workflow pattern, 16-9
sequential workflow with escalation
changing the escalation policy, 16-48
changing the expiration duration of the task when escalated, 16-48
changing the user to whom the task is escalated on expiration, 16-48
customizing, 16-48
definition, 16-9
implementation, 16-46
maximum number of times the task can be escalated, 16-48
pattern-specific parameters, 16-48
title of a user to whom to escalate the task, 16-48
use case, 16-47
workflow pattern, 16-9
service names
in adapters, 2-24
services
notifications, 2-23
overview, 2-22
sensors, 2-23
transformations, 2-23
workflows, 2-23
worklists, 2-23
setNodeValue extension function
description, G-36
short message service
See SMS, 2-23
simple workflow
changing the assignee after creating the user task, 16-32
changing the outcomes after creating the user task, 16-33
changing the user task to support multiple approvals, 16-33
customizing, 16-32
definition, 16-9
designed in JDeveloper BPEL Designer, 16-31
implementation, 16-31
use case, 16-32
workflow pattern, 16-9
simple workflow with automatic escalation
change the duration for the task assignment, 16-36
change the number of levels of escalation, 16-36
changing the title of the last user to whom the task is escalated, 16-36
changing the user to whom the task is escalated upon expiration, 16-37
customizing, 16-36
implementation, 16-33
maximum number of time the task can be escalated, 16-35
pattern-specific parameters, 16-35
title of user to whom to escalate the task, 16-35
use case, 16-35
simple workflow with automatic renewal
adding logic depending on number of times renewed, 16-40
changing the duration for the task assignment, 16-39
changing the number of levels of renewal, 16-40
customizing, 16-39
implementation, 16-37
maximum number of time the task can be renewed, 16-39
pattern-specific parameters, 16-39
use case, 16-39
simple workflow with escalation
definition, 16-9
workflow pattern, 16-9
simple workflow with renewal
definition, 16-9
workflow pattern, 16-9
single assignments, 16-20
SleepBroker
demos, 1-8
SMS
configuring wireless service provider

support, 15-13
notifications support, 15-12
SOAP interface
invoking a BPEL process, 12-6
SOAP services
invoking a BPEL process through, 12-2
performance issues when using Java code wrapped as a SOAP service, 10-2
using Java code wrapped as a SOAP service, 10-2
source code
reviewing in Eclipse BPEL Designer, 3-9
Source window
definition, 2-8
square-root extension function
description, G-36
starting
developer prompt, 2-3
Eclipse BPEL Designer, 2-3
JDeveloper BPEL Designer, 2-2
Oracle BPEL Console, 2-3
Oracle BPEL Process Manager components, 2-2
Oracle BPEL Server, 2-2
Oracle BPEL Worklist Application, 2-3
static assignments, 16-20
strings
concatenating, 4-8
converting to an XML element, 4-16
Structure window
definition, 2-11
location of in JDeveloper BPEL Designer, 2-5
submit more information
task action, 16-30
subtract-dayTimeDuration-from-dateTime extension function
description, G-37
suitcase JAR file, 3-17
definition, 19-6
location in a specific domain, 19-11
location of, 3-17
suspend
task actions, 16-30
switch activity
adding, 8-3
capabilities, C-15
in conditional branching logic, 8-2
references, 1-9
synchronous callbacks, 5-1
operational concepts, 5-6
required ports, 5-2
syncWaitTime attribute, 5-2
synchronous interactions
BPEL process as the client, 13-3
BPEL process as the service, 13-3
definition, 13-2
returning faults, 9-7
synchronous processes
reviewing the source code, 3-9
timeouts, 11-7
synchronous services
callbacks with the partner link and invoke activity, 5-2
calling, 5-6
demos, 1-8
example of invoking, 5-2
invoke activities, 5-5
invoking, 5-1
partnerLinkTypes, 5-4
port types, 5-4
ports, 5-2
tutorials, 1-11
syncMaxWaitTime property
increasing to prevent timeouts, 11-7
syncWaitTime attribute
in synchronous callbacks, 5-2

T

task actions
acquire, 16-30
change outcome for the task, 16-29
escalate, 16-30
in worklists, 17-2, 17-10
reassign, 16-30
release, 16-30
renew, 16-30
request more information, 16-30
request more information with reapproval, 16-30
resume, 16-30
route, 16-30
submit more information, 16-30
suspend, 16-30
update task details, 16-29
withdraw, 16-31
task continuation, 16-9, 16-56
changing a new workflow to an extended workflow, 16-58
customizing, 16-58
customizing applicable to the workflow pattern applies to the extended workflow also, 16-58
no pattern-specific parameters, 16-58
restrictions, 16-57
use case, 16-58
task field updates
in worklists, 17-24
task history
in worklists, 17-14
task notification variables
message variables, 16-62
task notifications
channels, 16-61
e-mail approval, 16-62
messages, 16-61
reminders, 16-63
task reassignment
in worklists, 17-22
TaskActionHandler component
definition, 16-4, B-3
WSDL operations, B-3
TaskManagementService component
definition, 16-4

TaskManager component
  definition, B-1, D-2
  integrating with a BPEL process, D-6
  integrating with Eclipse BPEL Designer, D-8
  WSDL operations, B-1
TaskRoutingService component
  definition, 16-5, B-2
  WSDL operations, B-2
tasks, 16-21
  acquired, 16-21
  actional e-mail account configuration, 16-74
  actions performed on, 16-29
  adding a task attachment from a business process, 16-26
  assigning a task to a user or role, 16-6, 16-19
  assigning dynamically with XPath expressions, 16-19
  assigning statically, 16-19
  assignment evaluations, 16-20
  assignments based on external services, 16-21
  attributes, 16-11
  autorelease duration configuration, 16-73
  creator, 16-12
  custom Worklist Application configuration, 16-74
  definition, 16-2
  delegations, 16-7
  dynamic assignments, 16-20
  escalations, 16-7
  expiration duration, 16-12
  expirations, 16-7
  flex fields, 16-15
  identification key, 16-12
  in worklist, 17-2
  notifications and reminders, 16-19, 16-60
  outcomes, 16-13
  payload, 16-11
  payload display, 16-12, 16-64, 16-71
  perform system actions, 16-30
  priority, 16-12
  resource bundles, 16-19
  restricted actions, 16-16
  routing, 16-7
  selecting users or group by browsing the user directory, 16-23
  sequential tasks, 16-20
  service configuration, 16-73
  setting task assignees from a dynamic delimited string, 16-22
  single assignments, 16-20
  static assignments, 16-20
  title, 16-11
  version-tracking attributes, 16-17
  workflow task attributes assigned using assign activities, 16-11
TCP tunneling
  definition, 6-8
terminate activity
  capabilities, C-16
  definition, 9-10
  fault handling, 9-10
  references, 1-10
third-party interactions
  definition, 13-10
throw activity
  capabilities, C-17
  references, 1-10
  throwing internal faults, 9-7
time
  assigning, 4-9
time duration format, 11-3
TimeOffRequestDemo
  demos, 1-8
timeouts
  designing with JDeveloper BPEL Designer, 11-4
  increasing the syncMaxWaitTime property, 11-7
  of BPEL processes, 11-2
  use case, 11-2
  using pick activities, 11-2
  using the wait activity, 11-4
  with synchronous processes, 11-7
timezone-from-dateTime extension function
  description, G-37
transform activity
  capabilities, C-17
  creating, 14-2
transformation functions
  Component Palette, 2-9
transformations
  adding XSLT constructs, 14-11
  auto mapping, 14-14
  auto mapping with confirmation, 14-16
  chaining functions, 14-9
  creating, 14-2
  creating an XSL map, 14-5
  definition, 2-23
  demos, 1-9
  dictionaries, 14-18
  editing functions, 14-8
  editing XPath expressions, 14-9
  functions, 14-7
  generating reports, 14-20
  in JDeveloper BPEL Designer, 2-23
  linking source target nodes, 14-6
  repeating elements, 14-13
  rules, 14-5
  setting constant values, 14-6
  testing the map file, 14-18
  tutorials, 1-11
  use case, 14-2
  user-defined functions, 14-9
  using arrays, 14-13
  using the XSLT Mapper, 14-5
  using XQuery and XSLT, 4-4
  XPath extension functions prefixed with xp20 or orcl, 14-7
  xsl choose conditional processing, 14-12
  xsl if conditional processing, 14-11
translateFromNative extension function
  description, G-38
translateToNative extension function

description, G-38
troubleshooting and workarounds
Oracle BPEL Worklist Application, A-3
sensors, A-1
tutorials
101.HelloWorld, 1-10
102.InvokingProcesses, 1-10
103.XMLDocuments, 1-10
104.SyncQuoteConsumer, 1-10
105.AsyncCompositeLoanBroker, 1-10
106.ParallelFlows, 1-10
107.Exceptions, 1-10
108.Timeouts, 1-10
109.CorrelationSets, 1-11
110.UserTasks, 1-11
111.CallingSessionBeans, 1-11
112.Arrays, 1-11
113.ABCARouting, 1-11
114.XSLTTransformations, 1-11
116.SendEmails, 1-11
117.ReceiveEmails, 1-11
118.JMSService, 1-11
119.JMSTopics, 1-11
120.XSQLExecution, 1-11
125.ReportsSchema, 1-11
126.DataAggregator, 1-11
127.OrderBookingTutorial, 1-11
128.GoogleFlow, 1-11
130.SendEmailWithAttachments, 1-11
701.LargeProcesses, 1-12
702.Bindings, 1-12
AQ adapter, 1-11
arrays, 1-11
asynchronous services, 1-11
batching, 1-11
BPEL exec extension, 1-11
conditional branching logic, 1-11
correlation sets, 1-11
database adapter, 1-11
debatching, 1-11
dynamic assignments in workflows, 16-21
EJB, 1-12
e-mails, 1-11
exceptions, 1-11
fault handling, 1-10, 1-11
file adapter, 1-11
file handler, 1-11
flow activity, 1-10
FTP adapter, 1-11
Hello World tutorial, 3-1
HTTP get method, 1-12
invoke activity, 1-10
Java method, 1-12
JMS, 1-11
JMS adapter, 1-11
JSP, 1-10
location of, 1-7
message service, 1-11
notifications, 1-11
parallel flows, 1-10, 1-11
pick activity, 1-10
receive activity, 1-10
reports, 1-11
RMI, 1-10
sensors, 1-11
synchronous services, 1-11
transformations, 1-11
user task activity, 1-11
workflow vacation request example, 16-98
workflows, 1-11
XML facades, 10-3
XML variables, 1-10
XSLT Mapper, 1-11
XSQL, 1-11

U

UDDI directories
browsing for services, 5-5
undeployment
of BPEL processes, 19-15
upper-case extension function
description, G-38
use with identity services, 16-2
use with LDAP, 16-2
user directory
selecting notification recipients by browsing the directory, 15-13
selecting users or groups in workflows, 16-23
user information
in worklists, 17-29
user properties
setting for notifications, 16-61
User Task 2.0 Macro
available for backwards compatibility only, 16-9, D-2
cannot be extended, 16-57
definition, 16-9, D-2
implementation, 16-56
task document fields, D-5
tasks cannot be viewed in the Oracle BPEL Worklist Application, 16-56
use case, D-2
workflow pattern, 16-9
user task activity
capabilities, C-18
creating, 16-31, 16-34, 16-38, 16-42, 16-51
tutorials, 1-11
userControls element, 16-81
user-properties.xml file
location of, 16-61
users
automatically created, 16-75
bpeladmin, 16-75
controlling access to in worklists, 17-39
creating, 16-77
default, 16-75
guest, 16-75
properties, 16-78
usersFunction

approver function, 16-96
using this guide, 1-5
utils
location of, 1-7

V

vacation request example
workflow tutorial, 16-98
VacationRequest
demos, 1-9
validation
when loading a process diagram, C-2
variable sensors
definition, 18-1
variables
complex type, 4-6
copying data between, 4-5
for task notification, 16-62
global, 3-10
initializing with expression constants, 4-4
initializing with literal XML, 4-4
versions
default, 19-15
definition, 19-3, 19-5
deployment of multiple versions of the same process, 19-3
do not overwrite existing versions of a process with newer versions in a production environment, 19-15
in production and development environments, 19-4
life cycles, 19-13
of BPEL processes, 19-3, 19-5
overwriting, 19-3
version-tracking attributes
code example, 16-17
in tasks, 16-17
visual flow
viewing the visual audit trail, 3-19
voice mail
configuring wireless service provider support for voice, 15-11
dynamically setting telephone numbers, 15-13
notifications support, 15-10

W

wait activity
capabilities, C-19
code example, 11-4
definition, 11-4
references, 1-10
Web interfaces
interacting with BPEL processes, 12-2
Web Service/SOAP interface
invoking a BPEL process, 12-6
WebLogic
JMS, 1-11
wf_config.xml file
location of, 16-62, 16-74
while activity
capabilities, C-20
in conditional branching logic, 8-4
references, 1-10
while loop
demos, 1-8
wireless service provider for SMS, 15-13
wireless service provider for voice, 15-11
withdraw
task action, 16-31
workflow patterns
adhoc workflow, 16-9, 16-54
extending, 16-9, 16-56
FYI task, 16-9, 16-55
parallel workflow, 16-9, 16-49
parallel workflow with final reviewer, 16-9, 16-52
restrictions on extending, 16-57
sequential workflow, 16-9, 16-40
sequential workflow with escalation, 16-9, 16-46
simple workflow, 16-9, 16-31
simple workflow with automatic escalation, 16-33
simple workflow with automatic renewal, 16-37
simple workflow with escalation, 16-9
simple workflow with renewal, 16-9
User Task 2.0 Macro, 16-9, 16-56
Workflow wizard
designing workflows, 16-10
workflows
actions performed on a task, 16-29
adding a task attachment from a business process, 16-26
approver function examples, 16-96
approver functions, 16-95
assigning a task to a user or role, 16-6
creating a user task activity, 16-34, 16-38, 16-42, 16-51
default JSP customizing, 16-67
definition, 2-23, 16-1
deleting, 16-11
demos, 1-8, 1-9
deploying the default JSP to OC4J, 16-71
design process, 16-10
designing in JDeveloper BPEL Designer, 16-10
editing, 16-11
extending patterns, 16-9, 16-56
features, 16-2
Identity service component, 16-5
in JDeveloper BPEL Designer, 2-23
JSP custom display, 16-72, 16-73
JSP default file customizing, 16-69
JSP default XML mapping file customizing, 16-67
JSP form, 16-65
multibyte payload in the task detail JSP, 16-71
notification, 16-60
definition, 16-2
notification service, 16-3
Notification service component, 16-5
notifications and reminders, 16-19

Oracle BPEL Worklist Application, 16-3
outcome-based modeling, 16-59
patterns, 16-8
patterns of flow, 16-7
procurement process example, 16-3
resource bundles, 16-19
restrictions on extending patterns, 16-57
selecting users or group by browsing the user directory, 16-23
storage of task history for auditing, 16-2
support for identity service, 16-2
support for JSP-based forms, 16-2
support for multiple workflow patterns, 16-2
support for task delegation, escalation, and reapproval, 16-2
support for task expiration and automatic renewal, 16-2
task
definition, 16-2
task assignment, 16-19
task assignment and routing
definition, 16-2
task assignment evaluations, 16-20
task assignments based on external services, 16-21
task continuation, 16-9, 16-56
task continuation restrictions, 16-57
task creator, 16-12
task expiration duration, 16-12
task flex fields, 16-15
task identification key, 16-12
task notifications, 16-60
task outcomes, 16-13
task payload, 16-11
task payload display, 16-12, 16-64
task priority, 16-12
task restricted actions, 16-16
task service configuration, 16-73
task title, 16-11
TaskActionHandler component, 16-4
TaskManagementService component, 16-4
TaskRoutingService component, 16-5
tasks acquired, 16-21
tutorials, 1-11
use cases, 16-6
vacation request example, 16-98
version-tracking attributes, 16-17
worklist
definition, 16-2
Worklist Application component, 16-5
XPath extension functions, 16-89
workflows using XSLT
task payload display, 16-71
Worklist Application component
definition, 16-5
worklists
accessing in local languages, 17-33
adding attachments, 17-20
adding comments, 17-19
advanced searches, 17-31
building a worklist application using the APIs, 17-40
building a worklist application using the remote APIs, 17-45
controlling access to information and actions for users, 17-39
customizing the Oracle BPEL Worklist Application, 17-34
definition, 2-23, 16-2
deleting attachments, 17-21
demos, 1-8
determining action permissions, 17-32
errors, 17-28
flex and task field updates, 17-24
in JDeveloper BPEL Designer, 2-23
logging into the Oracle BPEL Worklist Application, 17-4
Oracle BPEL Worklist Application, 17-4
Oracle BPEL Worklist Application action permission determination, 17-32
Oracle BPEL Worklist Application attachment adding, 17-20
Oracle BPEL Worklist Application attachment deleting, 17-21
Oracle BPEL Worklist Application attachment parallel tasks, 17-23
Oracle BPEL Worklist Application comments, 17-19
Oracle BPEL Worklist Application customizing, 17-34
Oracle BPEL Worklist Application errors, 17-28
Oracle BPEL Worklist Application flex and task field updates, 17-24
Oracle BPEL Worklist Application home page contents, 17-6
Oracle BPEL Worklist Application local language access, 17-33
Oracle BPEL Worklist Application payload viewing, 17-17
Oracle BPEL Worklist Application payload viewing of source XML, 17-18
Oracle BPEL Worklist Application request status, 17-26
Oracle BPEL Worklist Application requests for more information, 17-13
Oracle BPEL Worklist Application routing, 17-12
Oracle BPEL Worklist Application Task Actions list, 17-10
Oracle BPEL Worklist Application Task Details page contents, 17-9
Oracle BPEL Worklist Application task history, 17-14
Oracle BPEL Worklist Application task reassignment, 17-22
Oracle BPEL Worklist Application user and group information, 17-29
Oracle BPEL Worklist Application visual display of changes, 17-33
parallel tasks, 17-23
payload viewing, 17-17

payload viewing for source XML, 17-18
request status, 17-26
requesting more information, 17-13
routing, 17-12
setting task assignees from a dynamic delimited string, 16-22
task actions, 17-2, 17-10
task history, 17-14
task reassignment, 17-22
tasks, 17-2
use cases, 17-1
user and group information, 17-29
visual display of changes, 17-33
workspaces
creating in Eclipse BPEL Designer, 2-13
creating in JDeveloper BPEL Designer, 2-3
definition, 2-3
WS-Addressing, 6-6
definition, 6-6, 6-7
header includes correlation and callback address information, D-6
sending correlation IDs, 6-6
WSDL files
definition, 2-5
differences between document-literal styles and RPC styles, 4-1, 4-16
editing, 3-5
in Hello World tutorial, 3-4
modifying to generate a fault, 9-11
specifying when creating a partner link, 2-21
WSDL operations
Identity service component, B-3
Notification service component, B-2
TaskActionHandler component, B-3
TaskManager component, B-1
TaskRoutingService component, B-2
WSIF binding
definition, 10-2
for using Java code in a BPEL process, 10-2
WSIL directories
browsing for services, 5-5

X

XML data in BPEL, 4-2
XML documents
manipulating, 4-2, 4-4
overview, 4-2, 4-4
XML facades, D-14
definition, 10-3
generating with schemac, 10-3, 19-24
Java embedding, 10-3
tutorials, 10-3
XML schema
message types and variable types, 4-1
XML variables
tutorials, 1-10
XPath, 4-2
XPath expression functions
mimic XPath 2.0 standards, 16-89
XPath expressions
assigning numeric values, 4-6
assigning workflow tasks with, 16-19
dynamically creating another XPath expression, 4-12
dynamically setting e-mail addresses and telephone numbers, 15-13
editing in transformations, 14-9
entering Boolean expressions is switch activities, 8-4
examples, 4-3
fetching a data sequence element, 4-12
for notification messages, 16-61
getVariableData, 16-20
in conditional branching logic, 8-2
XPath extension functions, G-1
abs, G-1
addChildNode, G-2
add-dayTimeDuration-to-dateTime, G-1
addQuotes, G-2
appendToList, G-3
authenticate, G-3
batchProcessActive, G-3
batchProcessCompleted, G-4
clearTaskAssignees, G-4
compare, G-4
compare-ignore-case, G-5
copyList, G-5
countNodes, G-6
createDeliminatedString, G-7
create-delimited-string, G-6
create-nodeset-from-deliminated-string, G-7
current-date, G-7
current-dateTime, G-8
current-time, G-8
day-from-dateTime, G-8
doc, G-9
ends-with, G-9
format, G-9
formatDate, G-10
format-dateTime, G-10
format-string, G-10
genEmptyElem, G-11
generateGUID, G-11
generate-guid, G-11
getChildElement, G-13
getContentAsString, G-13
get-content-as-string, G-12
getConversationId, G-13
getCreator, G-14
getCurrentDate, G-14
getCurrentDateTime, G-14
getCurrentTime, G-14
getDomainId, G-15
getElement, G-15
getGroupIdsFromGroupAlias, G-15
getGroupProperty, G-16
getInstanceId, G-16
getLinkStatus, G-16
get-localized-string, G-12

getManager, G-17
getMessage, G-17
getNodes, G-18
getNodeValue, G-17
getNumberOfTaskApprovals, G-18
getPreference, G-18
getPreviousTaskApprover, G-18
getProcessId, G-19
getProcessOwnerId, G-19
getProcessURL, G-19
getProcessVersion, G-19
getReportees, G-20
getTaskAttachmentByIndex, G-20
getTaskAttachmentByName, G-20
getTaskAttachmentContents, G-21
getTaskAttachmentsCount, G-21
getTaskAutoReleaseDuration, G-21
getTaskReminderDuration, G-21
getUserAliasId, G-22
getUserIdsFromGroupAlias, G-22
getUserProperty, G-22
getUserRoles, G-23
getUsersInGroup, G-23
getVariableData, G-24
getVariableProperty, G-24
hours-from-dateTime, G-25
implicit-timezone, G-25
index-within-string, G-25
integer, G-26
isUserInRole, G-26
last-index-within-string, G-26
left-trim, G-27
listUsers, G-28
location of descriptions, 4-3
lookupGroup, G-29
lookup-table, G-28
lookupUser, G-27, G-29
lookup-xml, G-29
lower-case, G-30
matches, G-30
max-value-among-nodeset, G-30
mergeChildNodes, G-31
minutes-from-dateTime, G-32
min-value-among-nodeset, G-31
month-from-dateTime, G-32
ora
  getGroupProperty, 16-91
  getManager, 16-91
  getNumberOfTaskApprovals, 16-92
  getPreviousTaskApprover, 16-92
  getReportees, 16-92
  getTaskAttachmentByIndex, 16-93
  getTaskAttachmentByName, 16-94
  getTaskAttachmentContents, 16-94
  getTaskAttachmentsCount, 16-93
  getUserInRole, 16-92
  getUserProperty, 16-91
  getUserRoles, 16-92
  getUsersInGroup, 16-92
  lookupGroup, 16-90
  lookupUser, 16-89
orcl
  format-string, 16-95
  get-localized-string, 16-95
parseEscapedXML, G-32
prefixed with xp20 or orcl, 14-7
processXSLT, G-33
processXSQL, G-33
query-database, G-34
readFile, G-34
right-trim, G-34
search, G-35
seconds-from-dateTime, G-35
sequence-next-val, G-35
setNodeValue, G-36
square-root, G-36
subtract-dayTimeDuration-from-dateTime, G-37
timezone-from-dateTime, G-37
translateFromNative, G-38
translateToNative, G-38
upper-case, G-38
workflow related, 16-89
year-from-dateTime, G-39
XPath functions
  concat, 4-8
  countNodes, 4-12
  custom, 4-4
  dynamically setting e-mail addresses and telephone numbers, 15-13
  examples, 4-3
  getCurrentDate, 4-9
  getCurrentDateTIme, 4-9
  getCurrentTime, 4-9
  getElement, 4-14
  in transformations, 14-7
  indexing methods, 4-11
  mathematical calculations, 4-7
  mergeChildNodes, 4-15
  orcl
    get-localized-string(), 16-19
  parseEscapedXML, 4-16
  position, 4-11
  references, 1-9, 1-10
  selecting an data sequence element, 4-11
XPath queries
  copying data, 4-6
  examples, 4-3
  references, 1-10
XQuery, 3-15, 4-2, 4-4
xsl choose
  conditional processing, 14-12
xsl if
  conditional processing, 14-11
XSL map
  creating, 14-5
XSLMapper
  demos, 1-9
XSLT, 3-15, 4-2, 4-4
  Oracle BPEL Worklist Application payload display, 16-71

XSLT constructs
  adding in transformations, 14-11
XSLT Mapper
  adding XSLT constructs, 14-11
  auto mapping, 14-14
  auto mapping with confirmation, 14-16
  chaining functions, 14-9
  creating a map file, 14-3
  creating a transform activity, 14-2
  creating an XSL map, 14-5
  demos, 1-9
  dictionaries, 14-18
  editing functions, 14-8
  editing XPath expressions, 14-9
  functions, 14-7
  generating reports, 14-20
  layout in JDeveloper BPEL Designer, 14-3
  linking source and target nodes, 14-6
  repeating elements, 14-13
  rules, 14-5
  setting constant values, 14-6
  testing the map file, 14-18
  tutorials, 1-11
  use case, 14-2
  user-defined functions, 14-9
  using, 14-5
  using arrays, 14-13
  XPath extension functions prefixed with xp20 or orcl, 14-7
  xsl choose conditional processing, 14-12
  xsl if conditional processing, 14-11
XSQL
  tutorials, 1-11

Y

year-from-dateTime extension function
  description, G-39