

# **Oracle® Business Process Management**

Oracle BPM Studio Help

10g Release 3 (10.3.1)

January 2009

Copyright © 2006, 2008, 2009 Oracle. All rights reserved.

The Programs (which include both the software and documentation) contain proprietary information; they are provided under a license agreement containing restrictions on use and disclosure and are also protected by copyright, patent, and other intellectual and industrial property laws. Reverse engineering, disassembly, or decompilation of the Programs, except to the extent required to obtain interoperability with other independently created software or as specified by law, is prohibited.

The information contained in this document is subject to change without notice. If you find any problems in the documentation, please report them to us in writing. This document is not warranted to be error-free. Except as may be expressly permitted in your license agreement for these Programs, no part of these Programs may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose.

If the Programs are delivered to the United States Government or anyone licensing or using the Programs on behalf of the United States Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the Programs, including documentation and technical data, shall be subject to the licensing restrictions set forth in the applicable Oracle license agreement, and, to the extent applicable, the additional rights set forth in FAR 52.227-19, Commercial Computer Software--Restricted Rights (June 1987). Oracle USA, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

The Programs are not intended for use in any nuclear, aviation, mass transit, medical, or other inherently dangerous applications. It shall be the licensee's responsibility to take all appropriate fail-safe, backup, redundancy and other measures to ensure the safe use of such applications if the Programs are used for such purposes, and we disclaim liability for any damages caused by such use of the Programs.

Oracle, JD Edwards, PeopleSoft, and Siebel are registered trademarks of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

The Programs may provide links to Web sites and access to content, products, and services from third parties. Oracle is not responsible for the availability of, or any content provided on, third-party Web sites. You bear all risks associated with the use of such content. If you choose to purchase any products or services from a third party, the relationship is directly between you and the third party. Oracle is not responsible for: (a) the quality of third-party products or services; or (b) fulfilling any of the terms of the agreement with the third party, including delivery of products or services and warranty obligations related to purchased products or services. Oracle is not responsible for any loss or damage of any sort that you may incur from dealing with any third party.

# Contents

<b>Introduction.....</b>	<b>8</b>
Document Scope and Audience.....	8
Oracle Documentation and Resources.....	8
<b>Getting Started.....</b>	<b>10</b>
About Oracle BPM Studio.....	10
What's New in this version.....	11
Revision History.....	13
Version 6.0.....	13
Applying Product Updates.....	15
<b>Working with Studio.....</b>	<b>16</b>
Profiles.....	16
Studio Preferences.....	16
Setting Studio Preferences.....	16
Setting Project Preferences.....	16
Setting Engine Preferences.....	17
Setting Eclipse Preferences.....	17
Studio Preferences Reference.....	17
Views.....	24
Views Overview.....	24
Showing Views.....	25
Documentation View.....	25
Log Viewer View.....	26
Outline View.....	27
Problems View.....	27
Project Navigator View.....	28
Properties View.....	28
Simulation View.....	29
Variables View.....	29
Test Results View.....	30
Projects.....	30
Projects Overview.....	30
Oracle BPM Example Projects.....	31
Creating a Project.....	31
Importing a Project.....	32
Exporting a Project.....	32
Running a Project in Studio.....	33
Importing Designs.....	33
Creating a Project Report.....	34
Localization of Projects.....	34
Working with Source Control Systems.....	36
Setting Project Preferences.....	37

Project Properties Reference.....	37
Reusing Assets Across Projects.....	38
Processes.....	42
Business Process Overview.....	42
Creating a Process.....	43
Importing Designs.....	43
Setting Process Properties.....	44
Process Instance Overview.....	44
Defining the Layout for the Lanes in a Process.....	44
Process-Level Debugging.....	46
Creating a Process Simulation Model.....	48
Exposing a Process as a Web Service.....	48
Process Web Service Reference.....	49
Publishing a Process to AquaLogic Service Bus.....	49
Process Property Reference.....	50
Flow Objects.....	51
Flow Object Overview.....	51
Activities.....	52
Gateways.....	62
Events.....	68
Global Activities.....	75
Artifacts.....	79
Adding a Flow Object.....	80
Configuring a Flow Object Properties.....	81
Flow Objects Property Reference.....	81
Flow Object Icon Reference.....	95
Groups.....	97
Creating a Group.....	98
Groups and Transitions.....	98
Groups and Grab Activities.....	100
Group Properties.....	100
Flow Object Tasks.....	101
What is a Task?.....	101
Tasks Types.....	102
Transitions.....	110
Transitions Overview.....	110
Adding a Transition.....	111
Unconditional Transition.....	112
Conditional Transition.....	112
Business Rule Transitions.....	114
Due Transition.....	115
Exception Transition.....	118
Compensate Transition.....	119
Message Based Transitions.....	119
Variables.....	120

Creating Project and Instance Variables.....	120
Instance Variables.....	120
Predefined Variables.....	122
Project Variables.....	128
Local Variables.....	128
Screenflows.....	128
Screenflow Overview.....	128
Screenflow Timeout.....	130
Procedures.....	130
Procedures Overview.....	130
Creating a Procedure.....	132
Organizations.....	132
Organization Overview.....	132
Creating and Managing Organizations in Studio.....	135
Using Organizations with the Embedded Process Execution Engine.....	139
Attribute Reference.....	139
Simulations.....	142
Simulation Overview.....	142
Process Simulation Model.....	143
Project Simulation Models.....	144
Creating and Running a Process Simulation Model.....	145
Round-trip Simulations.....	146
Simulation Reference.....	147
Components Catalog.....	150
About Components.....	150
About the Components Catalog.....	151
Creating a Module.....	152
Deleting a Module.....	152
External Components.....	152
BPM Objects.....	189
BPM Object Overview.....	189
Creating a BPM Object.....	191
Attribute Overview.....	192
Defining an Attribute.....	192
Attribute Data Types.....	196
BPM Object Presentations.....	198
Creating a Presentation.....	198
External Resources.....	198
Creating an External Resource.....	199
External Resource Reference.....	199
Auditing.....	221
When Audit Events Are Generated.....	222
Which Audit Events are Generated.....	223
Configuring Auditing for a Process.....	223
Configuring Auditing Events for an Activity.....	224

Configuring the Generation of Audit Records for an Activity Group.....	224
Modifying the Generation of Audit Records for an Activity Group.....	225
<b>Advanced Use Cases.....</b>	<b>226</b>
Dynamic Business Rules.....	226
When to use Dynamic Business Rules.....	226
Using Dynamic Business Rules.....	227
Defining a Business Rule.....	228
Letting Participants Edit Business Rules.....	229
Handling Exceptions.....	230
Exception Handling in Oracle BPM.....	230
System Exceptions.....	231
Business Exceptions.....	231
Code-level Exception Handling.....	232
Process-level Exception Handling.....	233
Typical Exception Handling Flow.....	233
Creating an Exception Flow in a Process.....	234
Creating a Business Exception.....	234
Business Activity Monitoring (BAM).....	234
BAM Overview.....	235
Enabling and Configuring BAM in Studio.....	235
BAM Database.....	235
Using Variables in BAM.....	236
Creating a Predefined BAM Dashboard.....	237
Viewing BAM Dashboards in Studio.....	237
BAM Database Reference.....	237
Unit Testing BPM projects.....	243
Unit Test Overview.....	243
Creating a Unit Test.....	244
Running a Unit Test.....	244
Test Results View.....	244
Correlations.....	245
Correlation Sets.....	245
Defining a Correlation Set.....	245
Correlation Property Data Types.....	246
Correlations Example.....	246
End-User Interfaces on Oracle BPM.....	250
Building a User Interface.....	250
<b>Process Business Language (PBL).....</b>	<b>252</b>
PBL Overview.....	252
Language Basics.....	252
PBL Methods.....	252
Comments.....	252
Expressions.....	253
Programming Styles.....	254

Data Types.....	255
Variables.....	312
Operators.....	313
Statements.....	315
Regular Expressions.....	330
Programming.....	341
Objects.....	341
Code Conventions.....	344
Embedded SQL.....	359

# Introduction

This section provides general information about the *Oracle BPM Studio Guide*. This guide assumes that you have already installed Oracle BPM Studio. See the *Oracle BPM Installation Guide* for more information.

## Document Scope and Audience

This document is written for Business Analysts, Business Architects and Developers using Oracle BPM Studio for modeling and implementing business processes. It covers all the functionality provided by the BPM Studio workbench.

When reading this document from within Studio's help system, those sections not relevant to your current Profile are hidden.

If you are new to BPM Studio, we recommend you start with the *Oracle BPM Tutorial*, which gives you a practical step-by-step introduction to the product.

## Oracle Documentation and Resources

This section describes other documentation, resources, support, and training information provided by Oracle.

The table below lists a number of Oracle Documentation and Resources which will help you get started with Oracle BPM.

Resource	Description
Oracle BPM Documentation	The complete Oracle BPM 10.3 product documentation is available at <a href="http://download.oracle.com/docs/cd/E13154_01/bpm/docs65/index.html">http://download.oracle.com/docs/cd/E13154_01/bpm/docs65/index.html</a> .
Oracle BPM Product Page	The official BPM product page is available at <a href="http://www.oracle.com/technology/products/bpm/index.html">http://www.oracle.com/technology/products/bpm/index.html</a> and provides news, data sheets and useful links.
Oracle BPM Download Page	You can download the latest version of Oracle BPM from <a href="http://www.oracle.com/technology/software/products/ias/bea_main.html">http://www.oracle.com/technology/software/products/ias/bea_main.html</a> .
Online Help	<p>To access online help:</p> <ul style="list-style-type: none"> <li>• In BPM Studio, select <b>Help ► Help Contents</b> to access the complete Oracle BPM Studio help. Context help is also available by pressing the F1 key, or by selecting <b>Help ► Dynamic Help</b> from the menu.</li> <li>• In BPM WorkSpace, click on <b>Help</b> in the title bar, or click on the help icon (🔍) in the title bar of any panel for help about that panel.</li> </ul>
Oracle Technology Network (OTN)	The Oracle Technology Network features articles, blogs, and newsgroups which will help you make the most out of Oracle products. <a href="http://www.oracle.com/technology/index.html">http://www.oracle.com/technology/index.html</a>
User Groups	Visit the User Groups to collaborate with peers and view upcoming meetings. At Oracle forums: <a href="http://forums.oracle.com/forums/forum.jspa?forumID=560">http://forums.oracle.com/forums/forum.jspa?forumID=560</a>



Resource	Description
Technical Support	<p data-bbox="581 205 1463 275">If you cannot resolve an issue using the above resources, Oracle Technical Support is happy to assist.</p> <p data-bbox="581 283 1463 325"><a data-bbox="581 283 1040 325" href="http://www.oracle.com/support/index.html">http://www.oracle.com/support/index.html</a></p>

# Getting Started

---

The following topics provide general information that you may need to know before using Oracle BPM Studio. These sections include a general overview of Studio, what's new on this release and how to apply updates.

## About Oracle BPM Studio

Oracle BPM Studio is a desktop application that allows you to model and implement business processes. Studio creates a common interface for business analysts and developers by providing common views into the same process model.

Oracle BPM Studio allows you to integrate, design, test, and evolve your business activities using a process driven method to coordinate and manage internal and external business services.

### Process Design

Oracle BPM Studio provides a complete process modeling environment. Within an Oracle BPM project, you can create different process models that correspond to different areas of your business. This allows you to create models that account for all of the people, systems and organizations within your business. Each process contains activities, transitions, and roles that define the tasks and workflow.

In addition to the activities, transitions, and roles of your process, you can also create project variables that can be used to define Key Performance Indicators (KPI) for your business process.

Studio also allows you to comprehensively document how your process functions. Based on this documentation, developers can implement the process according to the specifications defined by the business analysts who created the process. This allows your process design to function as a contract between the process design and implementation stages.

After you have created a process model, Oracle BPM Studio allows you to run process simulations that mimic how your process behaves in production.

### Process Development

Oracle BPM Studio also provides a complete process development environment that allows you to go from the process modeling stage to a functioning production environment.

Oracle BPM Studio allows you to define the business rules and logic that ties your business process together.

For processes requiring integration with back-end applications, Oracle BPM processes communicate with these underlying application services through components. Components are also cataloged for use with the Oracle BPM adaptors framework that interfaces with application APIs. These APIs can be implemented in various technologies, including Java, EJB, COM, CORBA/IDL, JDBC/ODBC, XML, JMS and other middleware.

Technology adaptors connect to this standard technology instead of a particular application. This allows the component Catalog to connect to any object. It has the ability to introspect any object technology and read its methods and properties to create a 'wrapper' or 'proxies' that directly interfaces with it

Oracle BPM Studio also provides an environment for testing your business processes before they are deployed in a production environment.

### Profiles

To ensure that the appropriate level of functionality is presented to each type of users, Oracle BPM Studio provides different profiles based on each user type. See [Profiles](#) on page 16 for more information.

## Eclipse Framework

Oracle BPM Studio is built on the [Eclipse](#) platform.

























## What's New in Oracle BPM 10.3 Studio

This topic provides an overview of the main new features, improvements and changes in this release of Oracle BPM Studio.

### Standards Support

- By default, new processes now use horizontal swim-lanes. You can change the swim-lanes orientation individually for each process. You can define the default orientation for each project and for your Studio installation.
- BPM Studio now embraces the BPMN modeling elements and rendering constructs, and BPMN is the new default process diagram theme. Automatic activities and groups now support Loop conditions.

The new flow elements are categorized into Activities, Gateways, Events, Global Activities, Flow, Lanes and Artifacts. The name of some flow elements changed on this version.

<b>Activities</b> <ul style="list-style-type: none"> <li>•  Interactive</li> <li>•  Decision</li> <li>•  Automatic</li> <li>•  Group</li> <li>•  Subflow</li> <li>•  Process Creation</li> <li>•  Termination Wait</li> <li>•  Grab</li> </ul>	<b>Gateways</b> <ul style="list-style-type: none"> <li>•  Conditional</li> <li>•  Split</li> <li>•  OR Split (new)</li> <li>•  Multiple (old name: Split-N)</li> </ul>
<b>Events</b> <ul style="list-style-type: none"> <li>•  Message Wait (old name: Notification Wait)</li> <li>•  Send Message (old name: Process Notification)</li> <li>•  Timer (new)</li> <li>•  Compensate</li> </ul>	<b>Global Activities</b> <ul style="list-style-type: none"> <li>•  Global Creation</li> <li>•  Global Automatic</li> <li>•  Global Interactive (old name: Global)</li> </ul>
<b>Flow</b> <ul style="list-style-type: none"> <li>•  Connector</li> <li>•  Transition</li> </ul>	<b>Artifacts</b> <ul style="list-style-type: none"> <li>•  Measurement Mark</li> <li>•  Note</li> </ul>
<b>Lanes</b> <ul style="list-style-type: none"> <li>•  Lane (old name: Role)</li> </ul>	

- Studio is now built on top Eclipse 3.3. AquaLogic BPM 6.0 release was based on Eclipse 3.2.

## Workspace

- Workspace provides a new **edit mode** which allows users to change the configuration and layout of panels.
- Users of BPM Workspace can configure and save the layout of panels. A new tabbed interface allows you to define multiple pages, each with its own set of panels. You can export the layout configuration to an XML file and re-import it on a different environment or as a different user. Administrators can define layouts for all users in a certain Role.
- You can export the data in the Worklist panel to PDF (Portable Document Format) or CSV (Comma-Separated values).
- You can see a chart representation of the distribution of items in the Worklist panel.
- Workspace includes the following new panels:
  - **Task Panel:** Renders the execution of interactive tasks within the panel, instead of using the default modal dialogs.
  - **Dashboard Display Panel:** Provides a way to display Dashboards within a Panel.
  - **View Chart Panel:** Provides predefined graphical reports about process performance, work items distributions and workload.
  - **Application Panel:** This panel contains an application (the execution of a Global Interactive). Applications can respond to work item selections or run independently.
- The user can now do re-assignment operations on multiple instances at once.
- The Business Rules editor shows additional auditing information, including who and when a rule was modified.
- Workspace now (optionally) stores session-specific information as client-side cookies. This allows load-balancing on a cluster environment without affecting the user experience.
- This new version of Workspace provides a simplified and streamlined interface, focused on usability and ease of use.

## General

- Studio now supports Mac/OS 10.4 Tiger and Mac/OS 10.5 Leopard.
- Studio now supports Windows Vista.
- Studio now supports CVS and Subversion version control systems (VCS). Additional systems may work after installing their respective Eclipse VCS plugins but only CVS and Subversion are currently certified.
- The Studio UI incorporates Eclipse 3.3 improvements such as the following:
  - New Minimize/Maximize behavior: When minimizing view stacks in Studio, the view icons are placed on the nearest trim area. If a view is maximized, all other views are minimized, rather than hidden.
  - Tabs have a new color scheme based on your system title background color, and unselected tabs now also have rounded corners to match the appearance of selected tabs. When tabs become crowded, they now maintain their icon and no longer show an ellipsis in order to maximize the amount of useful information.
  - Firefox can now be set as the internal Web browser.

For more information on Eclipse 3.3 improvements, see [What's New in 3.3](#).

## Process Design

- Interactive tasks provide a new "**previewable**" property. The new Application Display Panel and Task Execution Panel of Workspace automatically start the execution of previewable tasks without locking the process instance. Enabled by default for Dashboards.
- New type of Activity: **Time Activity**. A process instance that arrives to this activity just sits idle until a timed event occurs.
- Option **Process Notification Immediately** on Termination Wait activities has been deprecated. Now both the Wait activity and the first activity in the interruption flow always execute in the same transaction.

- New auto-layout feature re-arranges all visual elements of a process diagram automatically, minimizing superpositions and aligning the flow as much as possible. Only available for processes with horizontal lane orientation.
- New process property (**Greedy Execution Mode**) indicates the Process Execution Engine to collapse contiguous automatic tasks in a single transaction. This mode of execution provides better performance for some processes. Disabled by default.
- A new Process-Level debugger allows developers to introduce breakpoints and debug complete processes running in Studio. When the execution reaches a breakpoint, the Engine pauses and Studio's debugging view appears. You can inspect variables, add new breakpoints, resume and continue execution.

## User Interface

- The **Business Analyst** and **Business Architect** profiles provide a simpler set of menu options and toolbars.
- New editor for BPM Object Presentations. It's easier to use, provides a true WYSIWYG interface (HTML-based), improved CSS support and a new Drag&Drop toolbar.
- You can now interrupt a running Simulation started with the **Run to the End** button.
- New **BETWEEN** operator added to Business Rules editor (on both Studio and Workspace). This operator works with Time and numeric types.
- The **Documentation** View now displays read-only documentation for the standard Fuego . \* components.
- New on-line help book **Oracle BPM Components Reference** provides reference documentation for the standard Fuego . \* components. Only available for the **developer** profile.
- This version introduces **Project Dependency**, which allows you to re-use components and role definitions from a common base project.

## Integration

- Oracle BPM now provides an extension to the Microsoft Office 2007 Ribbon. This extension allows users to submit documents to BPM processes right from the Office application.
- Added support for abstract types when cataloging XML Schemas.
- New timeout property added to external resources of type HTTP Server. Use this setting to control timeouts on web service invocations.
- Authentication information added to external resources of type JMS (Java Messaging System)
- Processes exposed as Web Services can now provide a `runProcess` operation, which synchronously executes the complete process (from begin to end). Only meaningful on fully automated processes.
- New component `Fuego.Social.ALIActivityStreamPublisher` provides operations to publish plain text messages to Oracle WebCenter Interaction (formerly AquaLogic Interaction) activity streams.

## Revision History

This section contains changes made in previous versions.

### Version 6.0

ALBPM 6.0 included the first implementation of Studio using the Eclipse platform, added support for several standards, and added built-in support of AquaLogic Service Bus and JDBC drivers. The Workspace Web application was re-built using modular components.

### Standards Support

- Process models in ALBPM are now compliant with the XPD 2.0 standard.
- Support for BPEL 2.0. You can import BPEL 2.0 models into an ALBPM Project, and new models can be designed within ALBPM Studio. The Process Execution Engine is now capable of executing BPEL 2.0 natively.

- ALBPM Studio application is now built on top of the Eclipse platform.

## Studio IDE

- Studio now includes a software agent for automatic problem reporting and feedback. In case of unexpected errors in Studio, an automatic report will be sent to Oracle for analysis. Studio will prompt you for approval before enabling this feature. We also encourage you to send us feedback using the **Help ► Feedback...** menu option.
- When you first start ALBPM Studio, you have to select one of the available profiles according your skill set: Business Analyst, Business Architect, Developer. ALBPM Studio presents a different subset of features depending on the selected profile. This keeps the user interface uncluttered, hiding what you don't need. All available features are visible under the Developer profile. The on-line documentation in Studio is also filtered depending on the active profile. To switch profiles go to [Help ► Welcome](#).
- This new release introduces the concept of Project Variables, replacing the External and Business Variables of previous versions. Project Variables are functionally equivalent to the old External Variables but are simpler to use: they are available to all processes in the project, with no need to "promote" them from External to Instance. When the new property **Business indicator** is enabled, Project Variables behave as the old Business Variables (they are used for BAM reporting).
- ALBPM project directories do not use the .fpr extension anymore.
- The Organization data and Simulation definitions are now accessed as nodes in the project tree.
- On previous version of Studio the Business Parameters of the project were accessible from the Variables panel on right. Now you access them from the Business Parameters node under the Organization node of the project tree.
- Integration with Version Control System feature (VCS) was re-implemented to leverage the Eclipse platform. This paves the way for supporting virtually any Source Control systems compatible with Eclipse.
- Each resource that is independently stored as part of an ALBPM Project is modified using an "Editor" tabbed panel, and you must explicitly save your changes on each resource with File > Save . For example, on earlier versions of Studio you add or modify a Participant using a separate dialog window. Now a special Participants editor opens in a new tab of the edition area. This makes it easier to work with Version Control systems, as each resource is managed and saved independently.
- Some editors may open nested editors (accessible via smaller tabs at the bottom of the editor). For example, the editor for Process models uses independent sub-tabs for the process diagram and for each opened process method.

## Process Designer

- You can now open several projects at the same time. Before opening a project, you first need to add it to your Studio workspace.
- Incremental compilation: There is no need for Publish&Deploy anymore. Once you start Studio's Process Execution Engine, the project is running. While it is running, the Execution Engine immediately applies changes you make to your project design and code.
- A new type of Interactive activity: Decision activities. This type of activity allows the end user to decide the next path a process instance will take (one of the possible outgoing transitions), based on the value of certain instance variables. The Process Execution Engine keeps track of those decisions over time and presents the end user with recommendations on what decision to take based on past experience.
- Business Rules: ALBPM Studio now provides a way of defining business rules using a graphical rules editor, without requiring any coding. After the project is deployed, authorized end users can also modify these rules on-the-fly, while the processes are executing. They can do so right from the ALBPM Workspace UI.
- Round-trip Simulation: You can now create Simulation models from the actual execution of the processes during a given period of time. This makes it easier to create realistic Simulation models.

## User Interface

- ALBPM WorkSpace has been re-designed and re-implemented from the ground up. It is based on a modern modular architecture which makes it easier to customize and integrate naturally with AquaLogic UI and WebLogic Portal. The old WorkSpace is still provided for backward compatibility but may be removed in future versions.
- Dashboards provide better quality graphics and end user interaction (i.e. rotation, detaching of pie sections).

## Integration

- Native integration with ALSB. You can now easily consume ALSB services from ALBPM and also register a business process in ALSB.
- Web Services in ALBPM now include support for WS-Security, Document-Literal style and WS-I compliance.
- ALBPM Studio now includes JDBC drivers for the most popular DBMS. This means you can integrate with Oracle, DB2 and Microsoft SQL Server right out of the box.
- PAPI has deprecated several methods in favor of new ones which follow a new naming convention. PAPI methods which were deprecated in ALBPM 5.7 have been deleted from the API.
- PAPI WebService 1.0 has been deprecated in favor of the new PAPI WebService 2.0. PAPI-WS 1.0 is accessible through ALBPM WorkSpace while PAPI-WS 2.0 is accessible through its own new Web Application (papiws). This new version is functionally equivalent to the native Java PAPI, and adheres to the WS-Security specification using the UserNameToken Profile implementation as well as HTTP Basic Authentication.


## Updating Studio

Follow this procedure to apply maintenance packs and patches to Oracle BPM Studio.

You initiate the Studio update procedure from within the Studio application.

Before updating Studio, check which version you currently have. On the main menu, click **Help ► About Oracle BPM Studio**.

To update Studio:

1. In Studio, save all your changes using **File ► Save All** .
2. In the main menu, click **File ► Updates > Studio Local Update**. Studio displays a dialog box indicating that Studio will close in order to execute the update, and asks if you want to continue.
3. Click **Yes**. Studio closes and the **Open** dialog box is displayed.
4. Specify the update (.upd) file and click **Open**. The update process runs. When the update procedure is complete, Studio will restart.
5. Verify that Studio is not at the expected revision by clicking **Help ► About Oracle BPM Studio**.

## Working with Studio

---

The following sections cover all the elements provided by the Oracle BPM Studio workbench.

### Profiles

Oracle BPM Studio provides three separate profiles. Each profile contains different levels of functionality which is targeted towards a specific user type.

Profile	Description
Business Analyst	Provides access to process modeling functionality, but does not contain any coding elements.
Business Architect	Provides access to process modeling functionality as well as modeling and service mapping, module definitions, and access to the BPM Object Catalog. Some basic coding functionality is supported.
Developer	Provides access to process modeling and all development functionality.

The currently selected profile is also used by the online help system of Oracle BPM Studio to hide content that is not relevant to your profile.


### Studio Preferences

Oracle BPM Studio provides different levels of customization preferences.

#### Setting Studio Preferences

Oracle BPM Studio preferences determine the general behavior of the Studio application.

To set Studio Preferences:


1. Ensure that the Project Navigator is visible.  
See [Showing Views](#) on page 25.
2. Select the  icon.  
The **Preferences** window appears.
3. Select a preference category from the list. The set of preferences for the selected category appears on the right.
4. Modify the preferences you need to modify  
For a detailed description of the available preferences, see [Studio Preferences Reference](#) on page 17.
5. Click **OK**.

#### Setting Project Preferences

Project preferences allow you to customize Oracle BPM Project-specific behavior.




To set project preferences:

1. Right-click on the Project whose preferences you want to edit.
2. Select **Project Preferences** () .
3. Edit the Project preferences.  
See [Project Properties Reference](#) on page 37 for detailed information on each Project preference.
4. Click **Ok**.

## Setting Engine Preferences

Engine preferences allow you to customize the Oracle BPM Process Execution Engine behavior. Engine preferences are saved independently for each BPM project.

1. Ensure that the Project Navigator View is visible.  
See [Showing Views](#) on page 25.
2. Right-click on the BPM Project whose Engine preferences you want to edit.
3. Select **Engine Preferences** () .  
The **Preferences** window appears.
4. Select a preference category from the list. The set of preferences for the selected category appears on the right.
5. Click **Ok**.

## Setting Eclipse Preferences

Eclipse preferences allow you to customize general behavior of the Eclipse Platform.

1. From the main menu: **Windows ► Preferences** .  
The **Preferences** window appears.
2. Select a preference category from the list. The set of preferences for the selected category appears on the right.
3. Click **Ok**.

## Studio Preferences Reference

The following reference provides detailed information about Studio Preferences.

### General

Options available on the General preference page.

Options	Description	Default
Overwrite application log at start time	Specifies if Studio clears the application log when you launch it.	On
Show hidden components	Specifies if the Component Catalog shows libraries added to the catalogue as dependencies of an introspected Java class.	Off
Default Lane Layout	Specifies the lane layout Studio uses when you create a new BPM Project. Possible values: <ul style="list-style-type: none"> <li>• Horizontal</li> <li>• Vertical</li> </ul>	Horizontal

**Activity**

Options available on the Activity preference page.

**General**

Option	Description	Default
Show properties automatically when adding a new object	Specifies if Studio displays a dialog with the activity properties immediately after you add a new activity.	On
Keep 'adding activity' mode	If selected the cursor does not return to its default state after adding an activity. This allows you add new activities of the type of selected.	Off
Show tooltips	Specifies if a tooltip with information about the activity appears when you roll the mouse over the activity.	Off
Layout automatically the design when adding objects	Specifies if Studio re-lays out the design when adding a new object to a process.	Off

**Messages**

Option	Description	Default
Show confirmation when replacing activity task	Specifies if Studio shows a confirmation message before replacing an activity task with the component you drag over the activity in the Structure View.	On
Ask to do layout automatically of the design when adding an activity	Specifies if Studio asks you if you want to re-layout the process design when you add a new activity. This option is enabled only when the property <b>Layout automatically the design when adding objects</b> is not selected.	On
On double click show	Specifies which information to show when double clicking on an activity. Possible values are: <ul style="list-style-type: none"> <li>• Main Task</li> <li>• Properties</li> </ul>	Properties
Show as title	Specifies the information used to identify the activity in the process diagram. Possible values are: <ul style="list-style-type: none"> <li>• Name</li> <li>• Description</li> </ul>	Name

**Transition**

Options available on the Transitions preference page.

**General**

Options	Description	Default
Show properties automatically when adding a new object	Specifies if Studio displays a dialog with the transition properties immediately after you add a new transition.	On
Ignore show properties option for unconditional transitions	Specifies if Studio ignores the value of the property <b>Show properties automatically when adding new</b>	On

Options	Description	Default
	<b>objects</b> when adding an unconditional transition. This property becomes available when you select the property <b>Show properties automatically when adding new objects</b> .	

### Visual Properties

Options	Description	Default
On Transitions Show	Specifies what type of information to show in the label of the transitions. If you choose <b>None</b> , no label is associated to the transitions. Possible values are: <ul style="list-style-type: none"> <li>• Condition</li> <li>• Name</li> <li>• Description</li> <li>• None</li> </ul>	Name

### Method Editor

Options available on the Method Editor preference page.

### Language

Options	Description	Default
Style	Specifies the PBL skin the method editor uses to display the code. Possible values are: <ul style="list-style-type: none"> <li>• PBL</li> <li>• Java</li> <li>• VisualBasic.NET</li> </ul>	PBL

### Auto Complete

Options	Description	Default
Auto Popup Members	Specifies if code completion is enabled.	On

### Others

Options	Description	Default
Allow Studio Debugger to commit transactions.	Specifies if the debugger commits SQL transactions. If you do not select this property, the debugger rolls back all SQL transactions before exiting.	On
Web Debugger Port	Specifies the port where the web server used for debugging runs. The Web Debugger uses this server to display messages in an embedded browser.	9595

### Printing

Options available on the Printing preference page.

**General**

Option	Description	Default
Complete page size with the last role	Specifies if the last role lane spreads through all the remaining space in the page.	Off

**Transitions**

Option	Description	Default
Print Conditional Transitions Information	Specifies if the information of conditional transitions is included when printing a process design.	On

**Reporting**

Options defined on the Reporting preferences page.

**General**

Option	Description	Default
Include use cases	Specifies if use cases are included when generating a Project Report.	On
Include implementation source code	Specifies if the source code of the implementation is included when generating a Project Report.	On
Include variables	Specifies if variables are included when generating a Project Report.	On

**Messages**

Options available on the Messages preference page.

**Process**

Option	Description	Default
Show confirmation when deleting a process	Specifies if Studio shows a confirmation message before deleting a process.	On
Show confirmation when deleting a folder	Specifies if Studio shows a confirmation message before deleting a folder.	On

**Method**

Option	Description	Default
Show confirmation box when deleting method	Specifies if Studio shows a confirmation message before deleting a method.	On

**External Resources**

Option	Description	Default
Show confirmation when Deleting External Resources	Specifies if Studio shows a confirmation message before deleting an External Resource.	On

## Variables

Option	Description	Default
Show confirmation when deleting variables	Specifies if Studio shows a confirmation message before deleting a variable.	On

## Log

Options available on the Log preference page.

### General

Option	Description	Default
Log Viewer Size		200 items
Update Frequency		30 seconds

## Connection Settings

Options available on the Connection Settings preference page.

### General

Option	Description	Default
Connection Settings	<p>Specifies the type of connection to the Internet. Possible values are:</p> <ul style="list-style-type: none"> <li>• Direct connection to the Internet</li> <li>• Manual proxy configuration</li> </ul> <p>If you select <b>Manual proxy configuration</b> you must specify the ID of the proxy server and its port.</p>	Direct connection to the Internet

## Presentation Preferences

Provides detailed information on the properties you can configure in the Presentation Editor.

### Presentation Preferences

The following table describes the preferences you can use to modify the appearance of the Presentation Editor.

Option	Description	Default
Use Preferences in Editor	Specifies if the Presentation Editor uses the defined <b>Presentation Preferences</b> when adding a new component to an existing presentation.	Off
Drop Target Background Color	Specifies the background color that identifies a drop target.	#C2FFBF
Drop Target Border Color	Specifies the color of the border that identifies a drop target.	#0F6AD9
Drop Target Hover Color	Specifies the color used to identify the currently selected drop target.	#52A4F4
Selection Border Color	<p>Specifies the color used to identify a selected component. Possible values are:</p> <ul style="list-style-type: none"> <li>• Custom</li> <li>• Default</li> </ul>	Default

Option	Description	Default
Cell Padding in Tables	If you select <b>Custom</b> , you can define which color to use using the color browser located next to this property.  Specifies the space the Presentation Editor uses to separate a component from the cell that contains it. The Presentation Editor uses this value to render tables, ignoring the cell padding defined for that specific table. You can use the preview to view the table using cell padding values you defined for your presentation.	4
Cell Spacing in Tables	Specifies the space the Presentation Editor uses to separate cells. The Presentation Editor uses this value to render tables, ignoring the cell spacing defined for that specific table. You can use the preview to view the table using cell spacing values you defined for your presentation.	4

### General

Option	Description	Default
Background Color	Specifies the background color of the presentation.	#FFFFFF
Layout	Specifies the layout of the presentation. Possible values are: <ul style="list-style-type: none"> <li>Table Default Layout</li> <li>Header/ Body/ Footer</li> </ul>	Header/ Body/ Footer

### Table

Option	Description	Default
Padding	Specifies the space that separates the component from the cell border.	7
Spacing	Specifies the space that separates the cells in a table.	0

### Cell

Option	Description	Default
Horizontal Alignment	Specifies the horizontal alignment of the component added to the cell. Possible values are: <ul style="list-style-type: none"> <li>Center justify</li> <li>Left justify</li> <li>Right justify</li> </ul>	Left justify
Vertical Alignment	Specifies the vertical alignment of the component added to the cell. Possible values are: <ul style="list-style-type: none"> <li>Top</li> <li>Center</li> <li>Bottom</li> </ul>	Center

**Text Field**

Option	Description	Default
Font Style	Specifies the font style used in text fields. Possible values are: <ul style="list-style-type: none"> <li>• Plain</li> <li>• Bold</li> <li>• Italic</li> <li>• Bold Italic</li> </ul>	Plain
Font Size	Specifies the font size used in text fields.	12
Maximum Column Count	Specifies the number of columns in text fields. The number of column determines the width of the text field.	20
Background Color	Specifies the background color used in text fields.	No color
Foreground Color	Specifies the foreground color used in text fields. The foreground color determines the color of the font.	No color
Horizontal Alignment	Specifies the horizontal alignment for the text in the text field. Possible values are: <ul style="list-style-type: none"> <li>• Center justify</li> <li>• Left justify</li> <li>• Right justify</li> </ul>	Left justify
Horizontal Alignment (numeric attributes)	Specifies the horizontal alignment for the numeric text in the text field. Possible values are: <ul style="list-style-type: none"> <li>• Center justify</li> <li>• Left justify</li> <li>• Right justify</li> </ul>	Right justify
Horizontal Alignment Within Cell	Specifies the horizontal alignment of the text field within the cell. Use this property to define a specific alignment for text field components. Possible values are: <ul style="list-style-type: none"> <li>• &lt;Cell Alignment&gt;</li> <li>• Center justify</li> <li>• Left justify</li> <li>• Right justify</li> </ul>	Left justify

**Drop-Down List**

Option	Description	Default
Horizontal Alignment	Specifies the horizontal alignment for the text in the drop-down list. Possible values are: <ul style="list-style-type: none"> <li>• Center justify</li> <li>• Left justify</li> <li>• Right justify</li> </ul>	Left justify

**Button**

Option	Description	Default
Transparent?	Specifies if the button uses a background color. If selected the background color defined for the button is ignored and the button allows you to see the component behind it.	On
Include Buttons	Specifies which predefined buttons to include when building a new presentation. You can include the following predefined buttons: <ul style="list-style-type: none"> <li>• Submit</li> <li>• Cancel</li> <li>• Reset</li> <li>• Refresh</li> </ul>	<ul style="list-style-type: none"> <li>• Submit</li> <li>• Cancel</li> </ul>
Layout Ordering	Defines how to layout the selected predefined buttons.	<ul style="list-style-type: none"> <li>• Submit</li> <li>• Cancel</li> </ul>

**Text**

Option	Description	Default
Horizontal Alignment Within Cell	Specifies the horizontal alignment of the text field within the cell. Use this property to define a specific alignment for text components Possible values are: <ul style="list-style-type: none"> <li>• &lt;Cell Alignment&gt;</li> <li>• Center justify</li> <li>• Left justify</li> <li>• Right justify</li> </ul>	Right justify
Capitalize first letter?	Specifies if the Presentation Wizard capitalizes the text component that displays the name of the attribute.	On
Separate names when underscore is found?	Specifies if the Presentation Wizard replaces underscores with blank spaces in the text component that displays the name of the attribute.	On
Capitalize each word?	Specifies if the Presentation Wizard capitalizes all the words in the text component that displays the name of the attribute.	On

## Views

### Views Overview

Views provide multiple ways of navigating resources within your Project.

#### Eclipse Standard Views

The Eclipse IDE provides a standard set of views that are available in all perspectives. For general information on using Views and for specific information about the default Eclipse views see the *Workbench User Guide*.



## Oracle BPM Custom Views

The Oracle BPM Perspective provides the following custom views:

View	Description
Documentation	Allows you to create and edit documentation for a Process and its Activities.
Log Viewer	Displays the error log for the embedded Process Execution Engine.
Problems	Displays information about errors and warnings that occur within a Project.
Project Navigator	Provides a hierarchical view of resources within a Project.
Properties	Displays the properties of a BPM Object Presentation.
Simulation	Allows you to run and view Process simulations.
Variables	Displays a list of variables grouped by type.
Test Results	Displays the results of CUnit and PUnit tests.



**Note:** Custom Views are only available from the Oracle BPM Perspective. They are not visible within other perspectives.

## Showing Views

This task outlines the procedures for showing Views in Oracle BPM Studio.

To show a view that is not visible within a perspective or to show a view that you have closed:

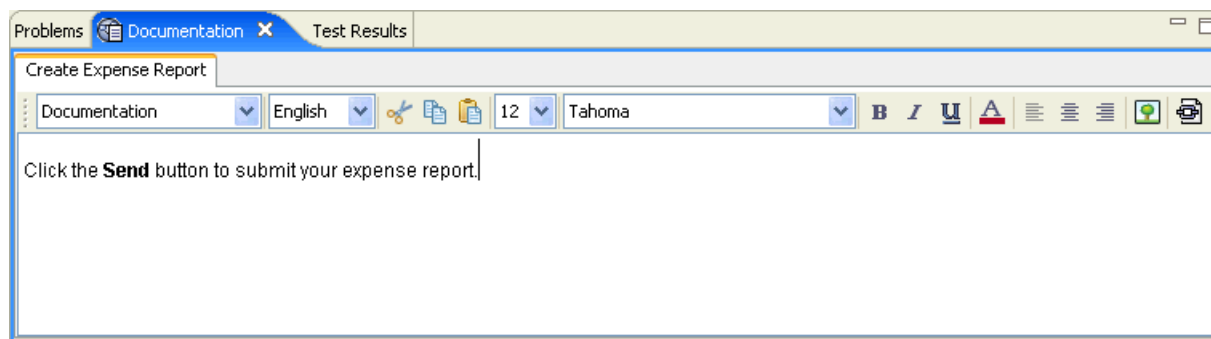
1. Select **Window ► Show View ► Other**.
2. Expand the folder of the type of View you want to open.
3. Select the View.

Views that are specific to the Oracle BPM Studio perspective are in the BPM folder.

## Documentation View

The Documentation View allows you to view, create and edit documentation for your Process and Activities.

This view provides a graphical text editor that allows you to perform basic text formatting, add images, and create hyperlinks.



Documentation Audience





Oracle BPM Studio allows you to create documentation for two difference audiences:

Audience	Description
Documentation	Provides information about a Project to end users. Content provided in this option appears in Workspace.
Use Case Documentation	Provides internal information about a Project that is useful to process architects and developers.

You can switch between audiences using the drop-down menu in the Documentation View toolbar. Both types of documentation appear in a generated Project Report.

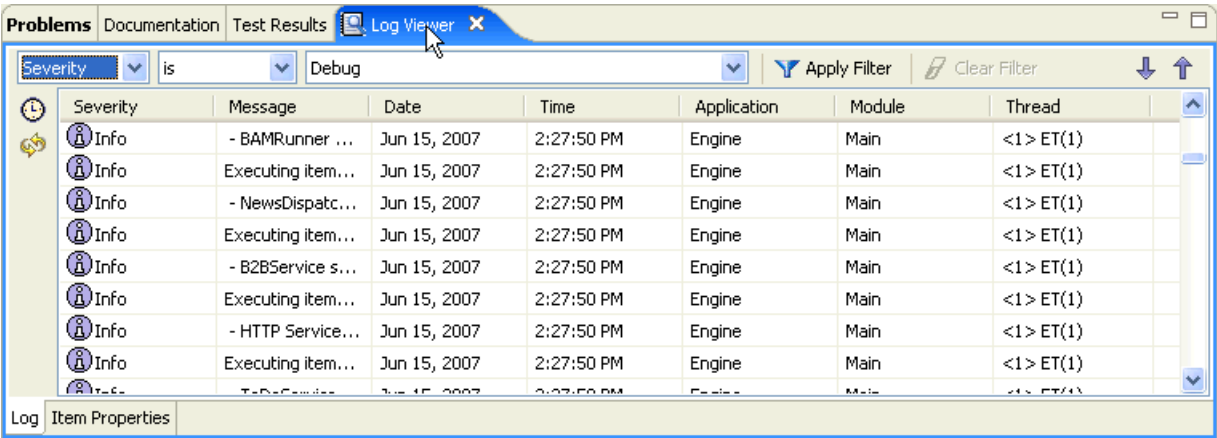
Documentation View Toolbar

The following table outlines the options available in the Documentation View toolbar:

Toolbar Element	Description
Audience Drop-down Menu	Selects the audience for the current content.
Language Drop-Down Menu	Specifies the language for the current audience.
	Cuts the current selection and copies it to the clipboard.
	Copies the current selection to the clipboard.
	Pastes the current selection at the cursor location.
Font Size	Defines the font size for the current selection.
Font Type	Defines the font type for the current selection. Available font types depend on the font styles installed on your system.
	Allows you to insert an image within the documentation.

Log Viewer View

The Log Viewer View provides logging information for the Embedded Process Execution Engine.



The Log Viewer only displays messages for the Project you use to start the Embedded Process Execution Engine. To view log messages for another Project, you must stop the engine and restart it using a different Project. See [Running a Project in Studio](#) on page 33.

### Logging Information

The Log Viewer displays the following information for each log message:

Column	Description
Severity	Indicates the kind of message (FATAL, SEVERE, WARNING, INFO, DEBUG).
Message	Contains the message that the Engine sends to the log.
Date	The time that the message was logged.
Time	The date the message was logged.
Application	Application that sent the message. All BPM system applications can send log messages to the log files.
Module	Module that sent the message.
Thread	Thread that sent the message.

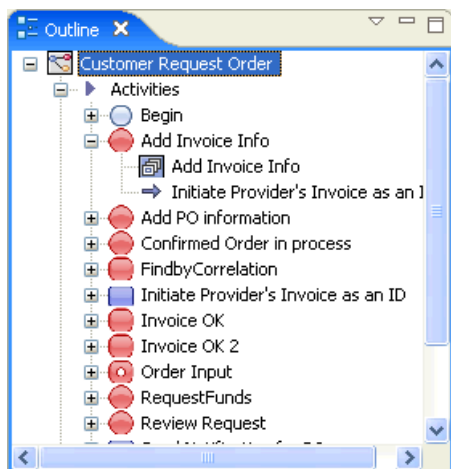
### Item Properties Tab

The Item Properties displays detailed information about specific log entries.

### Outline View

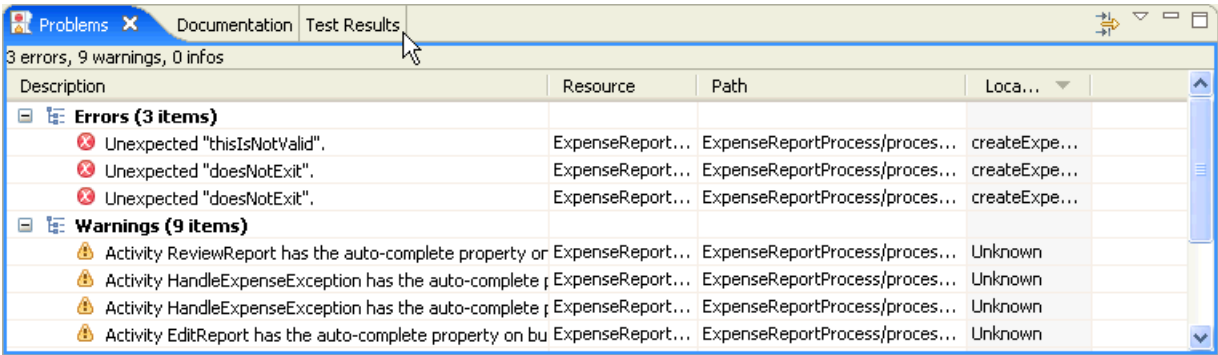
The Outline View displays an outline structure of a file that is currently open in an editor. The Outline View is a standard Eclipse View, but is frequently used within Oracle BPM Studio.

The contents of the Outline View depend on the contents of the currently highlighted editor. The following image shows an example of the Outline View of an Oracle BPM Process.



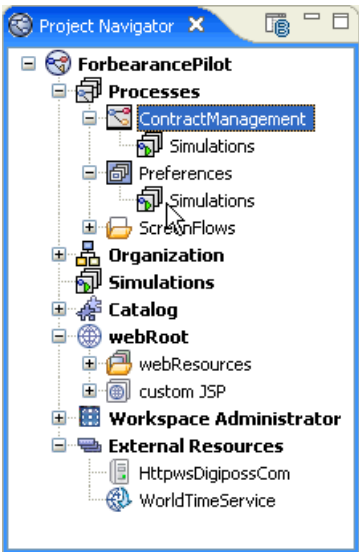
### Problems View

The Problems View displays information about errors and warnings that occur within the Project.




Project Navigator View

The Project Navigator View displays all of the Projects and project resources within your current workspace. The following figure shows a typical Oracle BPM Studio Project:

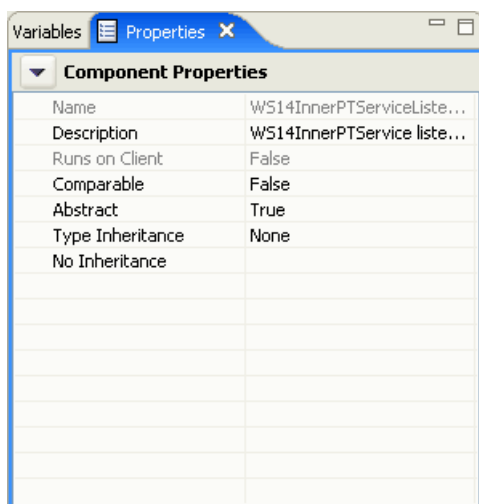


Setting Oracle BPM Studio Preferences

The  icon in the Project Navigator toolbar opens the Studio Preferences window. For more information on setting these preferences, see [Studio Preferences](#) on page 16.

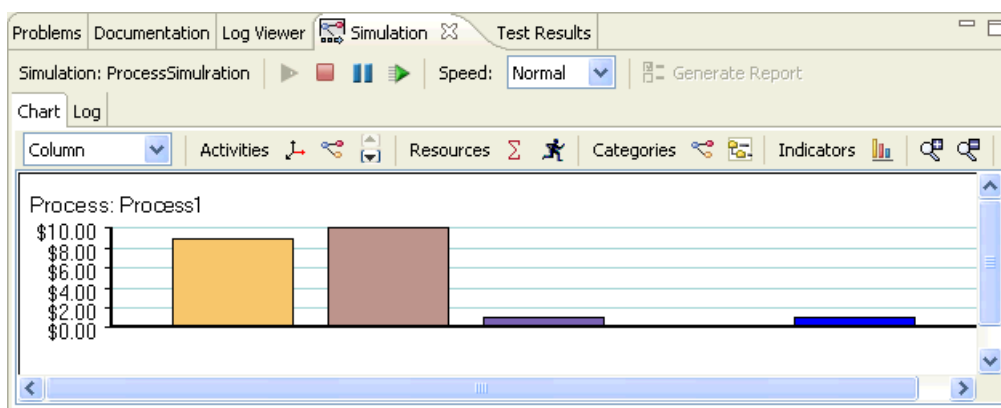
Properties View

The Properties View displays the properties of a BPM Object Presentation. This view is different from the standard Eclipse Properties view.



## Simulation View

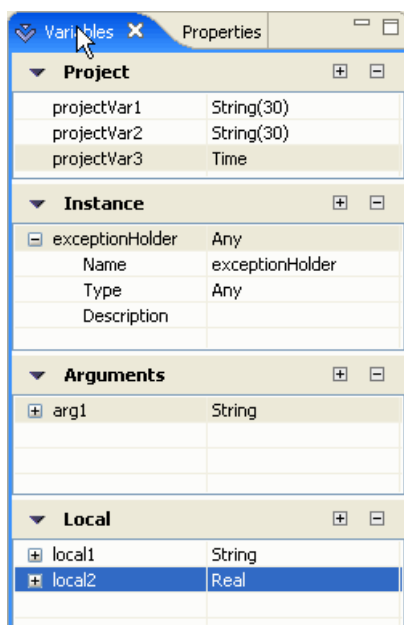
The Simulation View allows you to run and view Process simulations.



## Variables View

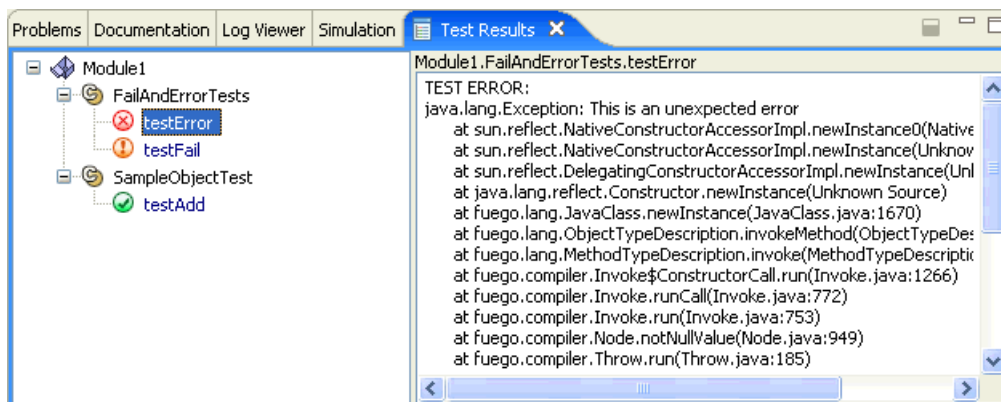
The Variables View displays the variables that are available within a Project.

The contents displayed in the Variable View depends on your current Profile and the context of the editor window you are viewing.



## Test Results View

The Test Results View displays results from PUnit and CUnit tests.



## Projects

Oracle BPM projects provide a way to organize, develop and manage different processes, users, components and systems catalogs. Projects allow you to collect and organize all of the processes and components associated with the business application you are developing. Projects developed in Oracle BPM Studio are published and deployed on Oracle BPM Enterprise.

## Projects Overview

A business project is the combination of a series of actions or operations pursuing a common business purpose. These activities, either human or automated, need to be executed in order to deliver a product or service. Business requirements may involve functional integration across the company or organization.

A Project involves not only the representation of all the elements that are part of a business, the human resources, the organization, the processes and the systems execution but also the way in which all of them interact.

Projects enable you to group processes that are related in some way and separate them from other groups.

Each project has its own component catalog so that you will be able to separate components used in some processes but not in others by grouping them in different projects. The project also contains all the abstract user roles used in it and its own Organization information required in order to deploy the project.

### Project Resources

Each Project contains the following resources that are visible in the Project Navigator:

Resource	Description
Processes	Contains the Processes, Procedures, and Screenflows defined for the Project. Within the Processes resource you can create multiple folders to organize resources.
Organization	Contains the Organizational elements that are defined for the Project.
Simulations	Contains simulations defined for the entire Project.
Catalog	Displays the list of catalog resources accessible from the Project.
webRoot	Contains user interface resources such as HTML pages and Java Server Pages.
Custom Views	Contains Views and Presentations that are defined locally for testing within Oracle BPM Studio.
External Resources	Contains connectivity information for external resource such as databases.

### Oracle BPM Example Projects

Oracle BPM Studio contains several example projects. These are grouped in sub-directories located under `<ORABPM_HOME>/samples`.

To view the sample projects, you can import them as an exported Oracle BPM Project. See [Importing a Project](#) on page 32.

The following table provides a brief description of each group of examples:

Sub-Directory	Description
advanced/	Projects showing non-trivial process models and advanced features.
basic/	Simple projects, using basic modeling features.
demos/	Complete self-contained demo projects with associated script.
integration/	Projects showing integration with external systems. Example: Ant scripts.
interop/	Projects showing interoperability with other Oracle products.

### Creating a Project

This section describes the basic procedures for creating an Oracle BPM Project.

To create a Project:

- 1. Select **File ► New ► BPM Project** .
- 2. Provide a Project name.
- 3. Choose a Project Root Directory.

The Project root directory contains all of the resources used by the Project.

- 4. Click **Next**.
- 5. Click **Finish**.


The new Project appears in the Project Navigator.

**Importing a Project**

Importing a project allows you to share Project resources.

To import a previously exported project:

- 1. Select **File ► Import...** .  
The **Import** window appears.
- 2. Select **BPM ► Exported BPM Project into Workspace** .
- 3. Click **Next**.
- 4. Click **Browse** to locate the Project you want to import.
- 5. Click **Open**, then click **Next**.
- 6. Optionally, change the name of the imported Project.

 **Note:** Click on the proposed project name to make it editable. The original name of the exported Project appears by default. However, if your workspace already contains a project with this name, you must specify a different one.

- 7. Click **Next**.

The Project files are expanded into your workspace.

- 8. Click **Next**, then click **Finish**.

The imported Project appears in the Project Navigator.

**Exporting a Project**

Exporting a Project allows you to create a self-contained, portable version of a Project.

To export a Project:

- 1. Right-click on the Project you want to export.
- 2. Select **Export Project**.
- 3. If your project depends on another project, select **Include base project**.  
For more information on Project Dependency, see [Reusing Assets Across Projects](#) on page 38.
- 4. Choose which libraries to include with the export file.

Option	Description
<b>Exclude all Libraries</b>	Does not include any of the libraries used in the project.
<b>Include all Libraries</b>	Includes versionable and non-versionable libraries. Use this option if you want to import the exported file to another installation of Studio.
<b>Include Versionable Libraries Only</b>	Includes only versionable libraries. Use this option if you want to use the exported file to publish and deploy a project in an Enterprise installation.



Option	Description
	It is not necessary to include these libraries because in an Enterprise installation you need to copy non-versionable libraries manually.

For more information on Library Versioning, see [Versionable Java Libraries](#) on page 169.

5. Click **Next**.
6. Provide the name and output directory of the exported Project file.
7. Click **Next**.
8. Click **Finish**.

## Running a Project in Studio

To run a BPM project in Oracle BPM studio you need to start the embedded Process Execution Engine.

To start the Engine in Studio:

1. In the Project Navigator, select the Project you want to use to start the Process Execution Engine.
2. Click the green play icon in the Eclipse toolbar.
3. Optionally, select from the following options:

Option	Description
<b>Delete Process Instances</b>	Clears all previous process instances before starting the Engine.
<b>Delete Log Files</b>	Clears all log files before starting the Engine.

4. Click **Ok**.  
The **Progress Information** window appears while the engine is starting. This may take several minutes.

When the Process Execution Engine starts, the green play icon changes to a red stop icon.

## Importing Designs

The Import Designs command allows you to import Processes, Screenflows and Procedures from other common Business Process Modeling formats.

To import designs:

1. In the **Project Navigator**, right-click on the process to which you want to add the imported process, or on the *Processes* node of that project.
2. From the context menu, select **Import Designs**.  
The **Import Designs** dialog box appears.
3. Specify the type of file you want to import in the **Files of type** drop-down list.

Option	Description
<b>Oracle BPM Process File (*.xpd1)</b>	Process definition file used for Oracle BPM processes. This is an XPDL 2.0 file with extensions.
<b>XPDL 1.0 Workflow Management Coalition Model (*.xpd1)</b>	Process Definition Language compliant with the 1.0 spec by the WfMC. Standard file type used to exchange process designs between process authoring tools. Because the standard allows for proprietary extensions to be included in an XPDL 1.0 file, the imported process can differ from the original version.
<b>Visio XML Drawings</b>	Process designs created with Microsoft Visio.
<b>Procedure Files (*.xad1)</b>	Oracle BPM procedure files.

Option	Description
Screenflow Files	Oracle BPM screenflow files.

- 4. Select the file you want to import.
- 5. Click **Open** .  
Each file you selected is imported to the project and opened in a new design editor window.

Creating a Project Report

Creating a project report allows you to view general and summary information about your project.  
To generate a Project Report:

- 1. Right-click on the Project you wish to create a report for.
- 2. Select **Project Report**.  
The **Report Options** window displays.
- 3. Choose the elements you want to include in the report:

Report Elements
Include Use Cases
Include Implementation Source Code
Include Variables

- 4. Click **OK**.
- 5. Select either *HTML* or *PDF* from the report output type drop-down list.
- 6. Select the folder where you want to output the report.
- 7. Click **OK**.

The report is generated. In PDF format, this will be a single file. In HTML, a folder and subfolders are created. You start reading an HTML report at the `index.html` page in the top folder.

Localization of Projects

Oracle BPM Studio allows you to localize elements of your business process. Most elements of a process that are visible within Workspace can be localized.

Supported Languages

When localizing a Project, you can add as many of the supported languages as necessary. The following languages are supported:

- Spanish
- Chinese (Traditional)
- Chinese (Simplified)
- Korean
- Japanese
- German
- French
- Italian
- Dutch
- Portuguese

English is the default language for a new Oracle BPM Project and is included automatically.

## Project Elements that Can Be Localized

After adding a languages to your project, you can add localized labels to the following aspects of your Project:

- Project Variables
- Business Rules
- Process Names
- Activity Names
- Input Tasks
- Decision Tasks
- Role Names
- View Names
- Presentations

## Adding a Language to a Project

Before you can localize different elements of your business processes, you must add a language to your project.

To add a language to your project:

1. Right-click on the Project where you want to add a language.
2. Select **Project Preferences**.
3. Click **Languages**
4. Click **Add**.
5. Select the language you want to add.  
See [Localization of Projects](#) on page 34 for a list of available languages.
6. Click **Ok**.

After adding a language to your Project you can localize individual elements of your processes.

## Localizing a Process Name

The following procedures show you how to localize a Project name. This localized name is used within Oracle BPM WorkSpace.

Ensure you have added a language to your process before localizing Activity names. See [Adding a Language to a Project](#) on page 35 for more information.

1. Right-click on the Project whose name you want to localize.
2. Select **Properties**.
3. Click the icon.
4. Enter the localized label next to the appropriate language.
5. Click **Ok**.
6. Click **Ok**.

## Localizing a Flow Object within a Process

The procedures outlined in this topic show you how to localize flow object names within your process. These localized names are used within Oracle BPM WorkSpace.

Ensure you have added a language to your process before localizing flow object names. See [Adding a Language to a Project](#) on page 35 for more information.

1. Open the Process containing the flow object you want to localize.
2. Right-click on the flow object.
3. Select **Properties**.

4. Select **Activity ID**.
5. Click the icon.
6. Enter the localized label next to the appropriate language.
7. Click **OK**.
8. Click **OK**.

## Working with Source Control Systems

The following topics describe how to share Oracle BPM projects and resources with your source control system.

### Source Control Overview

Source control systems enable you to share and control resources within your organization. Oracle BPM Studio allows you to easily integrate Project resources within your source control system.

Oracle BPM Studio uses a standard Eclipse layer for accessing source control system. Most source control systems provide plugins for Eclipse. Oracle BPM includes the CVS and Subversion plugins by default.

### Sharing Files Using Source Control

The following procedures show you how to share project resources using a source control system in Oracle BPM.

1. In the Project Navigator, right-click on the resource you want to share.
2. Select **Team ► Share Project**

The Share Project wizard appears. This wizard allows you to define connectivity information for your source control system.

3. To configure connectivity properties, follow the procedures defined in the *Team CVS Tutorial* or the *Team SVN Tutorial* in the *Getting Started* section of the *WorkBench User Guide*. The *WorkBench User Guide* is part of Eclipse standard documentation.

### Extracting Files from CVS Source Control System

The following procedure shows you how to access Oracle BPM Resources that are stored in a CVS source control system.

1. In the Project Navigator, select **File ► Import** .  
The **Import** wizard appears.
2. Expand **CVS**.  
The **Project from CVS** node appears.
3. Select **Projects from CVS**.
4. Click **Next**.  
The page to configure the connection preferences appears.
5. To configure CVS connection preferences, follow the procedures defined in the *Team CVS Tutorial* in the *Getting Started* section of the *WorkBench User Guide*. The *WorkBench User Guide* is part of Eclipse standard documentation.

### Extracting Files from Subversion Source Control System

The following procedures show you how to access Oracle BPM Resources that are stored in a Subversion source control system.

The following procedure assumes you defined a valid Subversion repository location. For information on how to configure Eclipse to work with a Subversion repository, see *Team SVN Tutorial* in the section *Getting Started* of the *WorkBench User Guide*.

To checkout an Oracle BPM project from a Subversion repository:


1. Choose **File ► Import** .  
The **Import** wizard appears.
2. Expand **Other**.  
The **Checkout Project from SVN** node appears.
3. Select **Checkout Projects from SVN**.
4. Click **Next**.  
The page to configure the connection preferences appears.
5. Select **Use existing repository location**.
6. Select a repository location.
7. Click **Next**.
8. Provide the credentials to log in to the selected repository.
9. Click **OK**.
10. Enter an author name and click **OK**.
11. Select an Oracle BPM project to check out.
12. Click **Finish**.

The selected BPM project appears in the Project Navigator View.

## Setting Project Preferences

Project preferences allow you to customize Oracle BPM Project-specific behavior.

To set project preferences:

1. Right-click on the Project whose preferences you want to edit.
2. Select **Project Preferences** ().
3. Edit the Project preferences.  
See [Project Properties Reference](#) on page 37 for detailed information on each Project preference.
4. Click **Ok**.

## Project Properties Reference

Project properties allow you to define different aspects of a BPM Project.

### General

Property	Description
Developing for J2SE Development	

### Languages

Property	Description
Current Language	Specifies the language used by this project. This setting is used to determine the language of Process names. If the localized version of a label is available, this version is displayed to users. If no localized version is available, the default language is used.

You can also use this page to add languages to a project. After adding a language to a project, you have the option of creating localized Activity labels.

## Exception Handling

Property	Description
Exception Handling	<p>Defines how exceptions are handled within the project. The following options are available:</p> <ul style="list-style-type: none"> <li>• Propagate: Causes un-handled exceptions are propagated to the parent or invoking process. The instance of the child process is aborted without executing the End Activity.</li> <li>• Handle Exceptions: Allows you to explicitly define how exceptions are handled.</li> </ul>
Automatically Generate Exception Handling Activity	Causes Studio to automatically generate an exception handling Activity and Role when creating a new process. This option is only available when <b>Handle Exceptions</b> is selected.

## Reusing Assets Across Projects

You can create a project with generic assets to use them in multiple projects. Then you can use the assets in the base project from another project. In this way you can leverage the effort made in other projects.

You can reuse the following assets:

- Processes
- Roles
- Components

To reuse an asset defined in another project, you need to configure your project to depend on that base project. For more information on how to do this, see [Configuring Project Dependency](#) on page 40.

A project can depend on only one project, however the project it depends on can also depend on another project.

Typically you create a base project with the most general assets. Some projects may directly depend on this project, so that they can use the assets defined in the base project. If there is a group of projects that share the need of other more specific assets, then you can configure them to depend on a project that contains those specific assets and in turn, depends on the base project.

## Reusing Processes

You can create a project with generic processes and use it as a library of processes. It should contain processes that you can use as templates to develop other processes.

Generic processes comply with the following:

- They define a main flow, leaving the alternative path to the specific uses of this template
- Their activities do not have an associated task
- Their process methods solve problems related to the process nature

To build your project library add new processes and roles, or copy already existing ones following the procedure described in [Copying a Process and its Roles to a Process Library](#) on page 40.

Your project should depend on the project containing the library of processes, so that you can use the processes in the base project as templates. To do this follow the procedures described in [Using a Process from a Process Library](#) on page 41.

The process you create using the template, uses the roles defined in the base project. If the activities in the process template have an associated task then it may use the components defined in the base project. However when you modify the process in the base project, your project does not reflect the changes.

## Reusing Roles

You can create a project with generic roles and use it as a library of roles. Typically you include the processes that use these roles in the same project.

To use the roles defined in the base project, your project should depend on it. For more information on how to use a role from the base project, see [Using a Role from a Role Library](#) on page 41.

Note that the roles in the base project are not copied to your project, your project uses them directly from the base project. When you modify a role in the base project, the depending projects reflect the changes.

## Reusing Components

You can create a project with reusable components and use it as a library of components. Typically you include the processes that use these components in the same project.

Another common practice is to create a project that only contains a library of components and make all the projects you create depend on this project so that they have access to those common components.

To build your component library you must catalogue the components or create new BPM Objects in the selected project. You can copy already existing BPM Object components by following the procedures described in [Copying a BPM Object Component to a Component Library](#) on page 41.

To use the components defined in the base project, your project should depend on it. For more information on how to use a component from the base project, see [Using a Component from a Component Library](#) on page 41.

Note that the components in the base project are not copied to your project, your project uses them directly from the base project. When you modify a component in the base project, the depending projects reflect the changes.

If the component in the base project uses an external resource, then when you use it from the dependent process it uses the external resource defined in the base project.

## Compiling a Project With Dependencies

To compile a project with dependencies you need to have access to both, the base project and the dependant project.

To compile a project that depends on another project, you need to add both projects Studio Workspace, and open them. If in turn the base project depends on another project you need to add it to your Studio WorkSpace and open it as well. If the base project is closed then you get errors when you compile your dependent project.

If someone else developed the project and shared it with you through a version control system, then make sure you checkout all the projects it depends on.

If you import a project that depends on other projects, the exported project file must include the dependencies. For details on how to export a project with dependencies, see [Exporting a Project With Dependencies](#) on page 42.

## Publishing a Process With Dependencies

This topic describes the considerations to take into account when publishing a project that depends on another project.

To publish a project that depends on another project in Studio, you need to add both projects to your Studio Workspace and open them.

To publish your project in an Enterprise environment you must use the **Exported Project** option for the publication source. To export the project and its dependencies follow the procedure described in [Exporting a Project With Dependencies](#) on page 42.

### Configuring Project Dependency

You can share assets between projects. The following procedure shows you how to configure your project to use the assets defined in a base project.

To use the assets of a project from another project, you have to open both of them in the current workspace.

To configure a project to depend on another project:

1. Right-click on the project.
2. Choose **Project Preferences**.
3. Select **Dependency**.
4. Select **Use project dependency**.
5. Select a project from the **Project base** drop-down list.

The following assets from the base project, are available in the selected project:

- Components
- Roles
- External Resources

You can copy processes from the base project to the selected project.

### Copying a Process and its Roles to a Process Library

The following procedure shows you how to copy an already existing process and its roles to the project you chose to use as a process library.

Before you follow this procedure make sure that the destination project does not depend on the project that contains your process, otherwise copying the process does not copy the roles.

To copy a process to the project that contains the process library:

1. Open the project that contains your project.
2. Open the project that contains your process library.
3. Copy the process to the destination project.

For information on how to do this, see [Copying a Process Between Projects](#) on page 40.

The selected process and the roles it uses are added to the project that contains the process library.

### Copying a Process Between Projects

The following procedure shows you how to copy a process from a project to another.

This procedure requires you to import the source and destination projects in your Studio Workspace and open them.

To copy a process from a project to another:

1. Select the process in the source project.
2. Drag the selected process to the destination project.

If the selected process uses screenflows, procedures, or sub-processes, you must copy them to the destination project. Otherwise the destination project does not compile.

The selected process and the roles it uses are copied to the destination project.



**Note:** If the destination project depends on the source project, then the roles are not copied to your project. Instead the copied process uses the roles directly from the source project.



### Using a Process from a Process Library

The following procedure shows you how to copy a process from the base project to use it as a template for new processes.

To use a process from a process library:

1. Configure your project to depend on the project that contains the process library.  
For information on how to do this, see [Configuring Project Dependency](#) on page 40.
2. Select the process you want to use as a template.
3. Drag the selected process to the project where you want to use it.
4. Drop the process in the processes node or in any folder underneath it.

The selected process is part of your project and you can modify it.



**Note:** If you modify the process in the base project, the copied process does not reflect the changes.

### Using a Role from a Role Library

You can reuse the roles defined in a project by configuring your project dependencies.

To use a role defined in a role library:

1. Configure your project to depend on the base project that contains the role library.  
For information on how to do this, see [Configuring Project Dependency](#) on page 40.
2. Right-click on the process in your project.
3. Choose **Add Role**.  
The **Role** window appears.
4. Choose a role from the list of roles.  
The list includes the roles from the base project as well as the roles from the current project.
5. Click **OK**.

The role you selected from the role library is added to your process.

### Copying a BPM Object Component to a Component Library

The following procedure shows you how to copy an already existing BPM Object component to the project you choose to use as a component library.

To copy a component to the project that contains the component library:

1. Open the project that contains your project
2. Export the BPM Object.  
Your BPM Object is exported to a zip file.
3. Add a module to the Component Catalogue.
4. Open the project that contains your component library.
5. Import the zip file that contains your BPM Object.

The BPM Object component is added to the project that contains the component library, in the module you created.

### Using a Component from a Component Library

You can reuse the components defined in a project by configuring your project dependencies.

To use a component defined in a component library, configure your project to depend on the base project that contains the component library. For information on how to do this, see [Configuring Project Dependency](#) on page 40.

The project depending on the base project, can use the components defined in the base project in the following contexts:

- The tasks of activities that require a component
- PBL methods
- Process instance variables
- Method arguments and argument mapping
- BPM Object variables
- BPM Object methods
- BPM Object inheritance

The components from the base project do not appear in the component catalogue of the project. They only appear when you browse the available types.

### Exporting a Project With Dependencies

The following procedure shows you how to include the project dependencies when you export a project.

To export a project with all its dependencies:

1. Right-click on the project you want to export.
2. Choose **Export Project**.
3. Select **Include base project**.
4. Click **Next**.
5. Enter a project name, in the **Project name** text field.
6. Select a location to store the exported project.
7. Click **Next**.  
A list of included files appears in the export wizard.
8. Click **Next**.  
A message to inform the export was completed successfully appears.
9. Click **Finish**.

The exported project file contains the selected project and its dependencies.

## Processes

### Business Process Overview

A business process is a sequence of business tasks and activities that, when executed, produces a well-defined outcome. Once this outcome is achieved, the process is complete.

A simple business process can involve hiring an employee, processing a sales order, or reimbursing a business expense. A more complex business process can involve many people and activities across an organization.

Sometimes the main goal of a process cannot be achieved. For example, if a product is out of stock, a shipping clerk may need to cancel a sales order. For this reason, a business process must provide for outcomes other than the principal goal. For example, if the product is out of stock it may be possible to offer the client an alternative that the client can then accept or reject. Thus, a process can have a range of possible outcomes.

### Activities

Business processes include logical steps, called *activities*, each of which can involve performing one or more *tasks*.

There are two types of activities: automatic and interactive. *Automatic activities* are executed automatically by the Process Execution Engine, whereas *interactive activities* require human input.

The activities of a business process are linked by *transitions*, which determine the order in which they are performed and the basic workflow of the process.

## Roles and Participants

Each interactive activity belongs to a *role*, that is, a title or job function performed by participants in the organization. For example, a role could be *Supervisor* or *Finance Administrator*.

*Participants* are the individuals who interact with the process. To perform an activity, a participant must be assigned the role that the activity belongs to. A participant can have one or more roles.

## Exceptions

Because it is often impossible to predict every outcome, a business process usually needs a way to deal with *exceptions*. An exception is an event in which a pre-defined outcome of a process cannot be reached.

The way in which a process deals with such an event, known as *exception handling*, can involve such steps as data clean-up or notifying a participant with a supervisory role that the situation needs attention.

## Creating a Process

Creating a new process allows you to begin modeling your business function.

Before performing the procedures in this task, ensure that you have created a Project. See [Creating a Project](#) on page 31.

To create a new Process:

1. In the Project Navigator, expand the Project where you want to create a new Process.
2. Right-click on **Processes ► New Process**
3. Provide a Process name and optional description.
4. Select the check box corresponding to how you want to generate Audit Trail Events for this Process.

See [Auditing](#) on page 221 for more information.

5. Click **OK**.

The new process is opened in a Process Editor window. All new processes are created with a Begin and End Activity connected by a default Transition. The new Process appears in the Processes resource in the Project Navigator.

## Importing Designs

The Import Designs command allows you to import Processes, Screenflows and Procedures from other common Business Process Modeling formats.

To import designs:

1. In the **Project Navigator**, right-click on the process to which you want to add the imported process, or on the *Processes* node of that project.
2. From the context menu, select **Import Designs**.  
The **Import Designs** dialog box appears.
3. Specify the type of file you want to import in the **Files of type** drop-down list.

Option	Description
<b>Oracle BPM Process File (*.xpd1)</b>	Process definition file used for Oracle BPM processes. This is an XPDL 2.0 file with extensions.

Option	Description
<b>XPDL 1.0 Workflow Management Coalition Model (*.xpd1)</b>	Process Definition Language compliant with the 1.0 spec by the WfMC. Standard file type used to exchange process designs between process authoring tools. Because the standard allows for proprietary extensions to be included in an XPDL 1.0 file, the imported process can differ from the original version.
<b>Visio XML Drawings</b>	Process designs created with Microsoft Visio.
<b>Procedure Files (*.xad1)</b>	Oracle BPM procedure files.
<b>Screenflow Files</b>	Oracle BPM screenflow files.

4. Select the file you want to import.
5. Click **Open** .  
Each file you selected is imported to the project and opened in a new design editor window.

## Setting Process Properties



## Process Instance Overview

A *business process* is a sequence of steps. A *process instance* is a specific item moving through those steps.

As the instance proceeds through the process, it is acted upon by various participants or processed automatically by software. For example, in a business process that handles purchases, each purchase order would be a process instance acted upon by such participants as shipping clerks, supervisors, and finance administrators.

Any number of instances can traverse a business process. For example, any number of purchase orders can traverse a business process that handles purchases.

Every instance has a specific history and properties. For example, a purchase order usually contains such data as a customer name, a list of items, an amount due, and dates of delivery and payment.

An instance can also have various status conditions. In the case of a purchase order, you want to know if it has been approved, billed, or paid, or if the requested products have been shipped.

Finally, each instance has a beginning and an end as defined in the business process.



**Note:** In order to understand what a business process *instance* is, you must first understand the concept of a [Business Process Overview](#) on page 42.



**Note:** In user interfaces designed for end users--for example, Oracle BPM WorkSpace--instances are also called *work items*.

## Defining the Layout for the Lanes in a Process

You can select how to layout the lanes in a process.

You can layout the lanes vertically or horizontally.

You can define how to layout lanes for:

- a single process
- new processes in a project
- new projects

## Defining the Layout for a Single Process

You can change the layout of a process at any given time. This change applies only to that process.

For more information on how to change a process lane layout, see [Changing the Process Lane Layout](#) on page 45.

## Defining the Layout for the New Processes in a Project

You can define which layout to use when creating a new process in a project. Note that the layout preference is defined by project and does not affect processes that already exist.

For more information on how to change the lane layout for the new processes in a project, see [Defining the Lane Layout for the New Processes in a Project](#) on page 45.

## Defining the Layout for New Projects

You can define the layout to use when creating a new project. The new projects you create use this preference to define their layout. For example if you prefer the horizontal layout, you might want to set this preference so that all the projects you create use horizontal layout for the new processes.

For more information on how to define the lane layout for new projects, see [Defining the Layout for New Projects](#) on page 45.

## Changing the Process Lane Layout

The following procedure shows you how to change the layout of the lanes in a process.

To change the layout of a process:

1. In the Project Navigator, right-click a process.
2. Choose **Properties**.
3. Select a **Lane Layout**.  
Options are:
  - Vertical
  - Horizontal
4. Click **OK**.
5. If the process editor is opened, click **Save**.

The layout of the lanes in the process changes to the selected layout.

## Defining the Lane Layout for the New Processes in a Project

The following procedure shows you how to configure the project preferences to use a certain layout the new processes in that project. By default this preference value is horizontal.

To select a lane layout for the new processes in a project:

1. Right-click on the project.
2. Select **Project Preferences**.
3. Select **Processes**.
4. Select a lane layout.
5. Click **OK**.

The new processes you create in this project use the selected lane layout.

## Defining the Layout for New Projects

The following procedure shows you how to configure Oracle BPM Studio to use a certain layout when creating new projects. By default this property is set to horizontal.

To set the project lane layout preference:

1. In the upper right corner of the **Project Navigator** tree, click **BPM Preferences** (⚙️).
2. Select **General**.
3. Select a lane layout.  
Options are:
  - Vertical
  - Horizontal
4. Click **OK**.

The next time you create a project, its process lane layout preference is the selected layout.

## Process-Level Debugging

You can debug the execution of a business process to detect the code that is causing the process unexpected behavior.

To debug a process you need to add breakpoints to signal the debugger where to temporarily suspend the process execution. You can add breakpoints at any script — whether it is a process method, a BPM object method, or a method within a screenflow. From any given breakpoint, you can then examine each subsequent line of code, add watchers, and use the debugging capabilities that the Eclipse platform provides.

### Adding a Breakpoint

The following procedure shows you how to add a breakpoint to your process.

To add a breakpoint:

1. In the **Process Navigator**, open the process you want to debug.
2. Double-click the activity where you suspect the problem may reside.  
You can only add breakpoints to activities with a method implementation.  
A code editor displaying the implementation for that activity appears.
3. Identify the line of code where you want to add a breakpoint, and right-click on the **marker bar**.



**Note:** Note: Oracle BPM does not support inserting breakpoints on empty lines or on lines with variable declarations.

4. Select **Toggle Breakpoint**.

A blue bullet appears at the point you selected.

### Configuring a Debugging Session

The following procedure shows you how to configure a debugging session.

To configure a debugging session:

1. Choose **Run ► Open Debug Dialog**.  
The **Debug** dialog appears.
2. From the toolbar, click **New launch configuration**.  
The right pane displays the debugging configurations.
3. In the **Name** field, type a name for the debugging session.



**Note:** Note that each session can debug only the project selected from the **Project Name** list. For any given project, you configure only one debugging session. To return to that session, select it from the **Debug** dialog.

4. From the **Project Name** list, select the project you want to debug.
5. Click **Apply**.
6. If you want to start debugging your process, click **Debug**, otherwise click **Close**.

### Debugging a Process

The following procedure shows you how to debug a process.

To debug a process:

1. Choose **Run ► Open Debug Dialog**.
2. Select a debug configuration.
3. Click **Debug**.
4. If the engine is not running, a dialog to start it appears. Click **OK** to start the engine.
5. Choose **Run ► Launch Workspace**.  
This launches your browser. A Workspace dialog prompts you to log in.
6. Log in as a participant with sufficient roles and privileges to execute the portion of the process you are testing.  
For example, if you inserted a breakpoint in or after an interactive activity, then log in as a participant who can execute that activity.
7. Create a process instance and move it through the process until it reaches the activity in which you previously inserted a breakpoint in Studio.
8. Return to Studio.

Notice the following:

- The currently executing line is highlighted in the code editor and a small arrow on the left shows the current code pointer.
  - The **Debug View** displays the current debugging session and call stack.
  - The **Variables View** displays the variables available in the selected method in the call stack.
9. Debug your process using the actions in the **Debug View**. For information about the available actions, see [Debugging Actions](#) on page 47.

### Debugging Actions

The following reference describes the available actions when debugging a process.

You can run these actions either by clicking on the **Debug View** toolbar, or by selecting them from the **Run** menu.

Action	Icon	Description	Shortcut
Resume		Resumes the process.	F8
Terminate		Finishes the debugging session.	Ctrl + F2
Disconnect		Finishes the debugging session.	
Step Into		Steps into the current highlighted methods, allowing you to debug the statements it contains.	F5
Step Over		Runs the currently highlighted statement without stepping into it.	F6
Step Return		Steps out of the current method.	F7

### Correcting a Process

Once you identify where the problem lies in the code, you can correct it by following these steps.

To correct the problem:

1. Stop debugging by clicking **Disconnect** in the **Debug View** toolbar.
2. Correct the process.
  - a) In the code editor, correct the problem.
  - b) Choose **File ► Save**.
  - c) Click **Reload**.
3. Choose **Run ► Launch Workspace**.
4. Log into WorkSpace as a participant with the roles and privileges needed to execute the activity you have corrected.
5. Verify your corrections by running the activity.

### Creating a Process Simulation Model

The following steps describe how to create a Process Simulation Model.

1. In the Project Navigator View, expand the Project where you want to create the Process Simulation Model.
2. Expand **Processes**.
3. Right-click on the Process.
4. Select **New Process Simulation Model**.
5. Enter a name for your new Process Simulation Model.
6. Click **OK**.

The Process Simulation Model appears in the editor window. It also appears as a Resource in the Project Navigator View.

You can define the behavior of your Process Simulation Model.

### Exposing a Process as a Web Service

Exposing a Process as a Web Service allows it to be accessed externally by other applications within your enterprise.

Before performing the procedures in this task, ensure that you have created a Process. See [Creating a Process](#) on page 43.

1. In the Project Navigator, expand the Project containing the Process you are exposing as a Web Service.
2. Expand **Processes**.
3. Right-click on your Process, then select **Process Web Service**

The Process Web Services tab is displayed in the Process Editor.

4. Click **Add** to create a new Web Service operation.
5. Provide the required information for each field.  
See [Process Web Service Reference](#) on page 49 for more information.
6. Click **Ok**.

The new Web Service operation appears in the table. You can add more operations as required.

7. Configure the optional **Advanced Options**.  
See [Process Web Service Reference](#) on page 49 for more information.



## 8. Select **File ► Save**

The process is exposed as a Web Service. If you start the Process Execution Engine using this process, you can view the deployed Web Service at: <http://localhost:9000/>. See [Running a Project in Studio](#) on page 33 for information.

## Process Web Service Reference

The properties listed in these topics are available when exposing an Oracle BPM Process as a Web Service.

### Operations

Property	Description
Operation Name	Defines the name used to refer to the operation. This name will become the method name to execute this operation.
Operation Type	Specifies the operation type: <ul style="list-style-type: none"> <li>Process Creation if this operation is to create an instance.</li> <li>Process Notification if this operation is to notify an instance waiting in a Notification Wait activity.</li> </ul>
Activity	<ul style="list-style-type: none"> <li>if the operation is for Creation then the Begin activity is automatically selected.</li> <li>if the operation is for Notification operations then select the Message Wait event in which the instance is waiting and to be notified using this operation. The Wait Activities receiving notifications be marked as Receiving External Events</li> </ul>
Argument Set Name	Determines the argument set the operation uses to receive arguments. This allows you to define more than one operation of the same type, each one receiving different arguments. If it is a Notification operation then you can define to use a Correlation.
Use Correlation	
Argument / Type	Displays the argument variables and types defined in the Argument Set Name.

## Publishing a Process to AquaLogic Service Bus

Oracle BPM Processes can be used within AquaLogic Service Bus.

Before performing the procedures in this task, you should ensure that AquaLogic Service bus is configured and running.

To publish an Oracle BPM process to AquaLogic Service Bus.

### 1. Create a Process.

See [Creating a Process](#) on page 43 for more information.

### 2. Expose the Process as a Web Service.

See [Exposing a Process as a Web Service](#) on page 48.

### 3. Run the project. This starts the Process Execution Engine.

See [Running a Project in Studio](#) on page 33.

### 4. Create an ALSB External Resource of type Management Host.

See [Creating an External Resource](#) on page 199 and [AquaLogic Service Bus](#) on page 215 for more information.

### 5. Create a second ALSB External Resource.

- a) Configure the External Resource as described in [Creating an External Resource](#) on page 199.
- b) Set the type to Process Registration.

This can be the same External Resource you configured in the previous step.

- c) Provide a project name.

This is the name of the ALSB project where the process will be published to. This can be a new or existing project.

- d) After you have provided all of the information, click **Create Structure**.

The AquaLogic Service Bus project is updated or created.

6. Register the End Point

- a) Right-click on the Project you used to start the Process Execution Engine.
- b) Select **Register End Point**.

The **AquaLogic Service Bus Registration** window appears.

- c) Select the Registration Configuration you want to use.

This can be the ALSB Process Registration External Resource you created earlier.

- d) Select Yes in the first column of the process you want to register.

- e) Click **Register**.

A message is written to the log window. The status of the process is changed to **up to date**. You can publish as many processes as necessary.

The process you registered is visible in the AquaLogic Service Bus Project Explorer under Business Services.

Process Property Reference

Process properties define specific attributes and behavior of a process.

General Properties

The following properties can be configured for a process under the **General** tab:

Property	Description
Name	Specifies the name of the Process. You can localize the name of the process for different languages.
Id	Specifies the ID of the process. This value is defined when the process is created and cannot be modified.
Description	Provides a description of the process. This is used when generating project documentation. You can localize the description of the process for different languages.
Variation	Specifies an additional label to characterize the process. Use it to distinguish between two processes which have the same name, indicating that one is a <i>variation</i> of the other . The Project Navigator displays the variation in parenthesis next to the Process name. This value is defined when the process is created and cannot be modified.
Author	Specifies who designed the Process.
Lane Layout	Specifies how to layout the lanes in the process.

## Advanced Properties

The following properties can be configured for a process under the **Advanced** tab:

Property	Description
Generate Events	Defines how Auditing Events are generated for the Process. See <a href="#">When Audit Events Are Generated</a> on page 222 for details.
Greedy Execution Mode	Defines whether the Process Execution Engine executes each automatic activity in a separate transaction ( <b>Disable Greedy Execution</b> ) or if it executes consecutive automatic activities into the same transaction ( <b>Enable Greedy Execution</b> ). By default, greedy execution mode is disabled.

## Flow Objects

This section provided detailed information about the different flow objects you can use to model a process.

### Flow Object Overview

A flow object models a step in a business process.

The following table describes different categories of flow objects:

Category	Description	Flow Object
Activity	Activities represent the work that companies perform.	<ul style="list-style-type: none"> <li>• Interactive</li> <li>• Decision</li> <li>• Automatic</li> <li>• Group</li> <li>• Subflow</li> <li>• Process Creation</li> <li>• Termination Wait</li> <li>• Grab</li> </ul>
Gateway	Gateways control the divergence and convergence of the process flow. They determine if the paths branch, fork, merge or join to other paths.	<ul style="list-style-type: none"> <li>• Conditional</li> <li>• Split</li> <li>• Or-Split</li> <li>• Multiple</li> </ul>
Event	Events affect the process flow. They happen during the course of a business process. They generally have a cause and an impact.	<ul style="list-style-type: none"> <li>• Message Wait</li> <li>• Send Message</li> <li>• Timer</li> <li>• Compensate</li> </ul>
Global Activity	Global activities handle global requirements that are not associated with a specific process instance.	<ul style="list-style-type: none"> <li>• Global Creation</li> <li>• Global Automatic</li> <li>• Global Interactive</li> </ul>

Category	Description	Flow Object
Artifact	Artifacts provide additional information about the process.	<ul style="list-style-type: none"> <li>Measurement Mark</li> </ul>

## Activities

This section provides information about the activities you can use to design a process.

### Interactive Activity

Interactive activities allow you to model tasks that require human involvement.

You can use Interactive activities to:

- present the user forms to collect data.
- allow the user to assign a value to instance and predefined variables.
- display information.

### Adding an Interactive Activity

When you add an Interactive activity, you should take into account the following considerations.

#### Roles

You must add Interactive activities within a role lane in the process. When you try to add an Interactive activity outside a role lane, Studio prompts you for an already existing role or allows you to create a new one.

The role lane where you add the Interactive activity determines which users can access it. Only the users assigned to that role can run the Interactive activity.

#### Transitions

You must connect an Interactive activity to the process flow using an inbound and an outbound transition. An error appears in the **Problems View** if the inbound or outbound transitions are missing.

#### Variables

The task associated to an Interactive activity can access the following types of variables:

- predefined variables
- instance variables
- arguments
- local variables

Some tasks may require you to define an argument mapping. For more information, see the corresponding task in [Flow Object Tasks](#) on page 101.

### Using Interactive Activities as an Exception Handler

You can use an Interactive activity to handle a business exception. The activity that throws a user-defined exception is connected to the Interactive activity through an exception transition. For more details about exception handling, see [Handling Exceptions](#) on page 230.

### Defining the Task of an Interactive Activity

You must define the logic of an Interactive activity in the main task. If appropriate you might define optional tasks to help the user run the main task.

There are different options to implement the tasks of an Interactive activity. Each option addresses specific types of interaction.

The tasks of an Interactive activity can have the following implementation types:

- Method
- Component
- Screenflow
- External
- Input
- Display

The default implementation type is method.

For more information about the different implementation types, see [Flow Object Tasks](#) on page 101.

## Optional Tasks

You can run optional tasks while you are running the main task of an Interactive activity. These tasks are usually related to the main task. They provide information that you might need to complete the Interactive activity.

Optional tasks are repeatable and read-only. They are not allowed to modify the instance information because this information is used by the main task.

### Viewing an Interactive Activity in Workspace

Describes how Workspace displays Interactive activities.

You can view and process Interactive activities in Workspace if you have the required role.

According to the Workspace configuration Interactive activities display in one of the following ways:

- A dialog
- A maximized dialog
- A pop-up window
- Within the detail panel

You can search for instances sitting in an Interactive activity or define a view that includes them. To view the instance details select the instance in the search result or in the view.

You can view information about the processing of an Interactive activity in the audit trail.

### Running an Interactive Activity

Describes how the Process Execution Engine processes an Interactive activity.

Generally Interactive activities require human involvement to complete their task. Only after the user completes the Interactive task associated to the activity, the instance moves to the next flow object in the process flow.

The Engine processes Interactive activities using interactive threads. If the implementation of the Interactive activity is of type screenflow, then the engine releases the interactive thread at the beginning of the screenflow and obtains a new interactive thread when the user completes the screenflow. When possible you should use screenflows to implement Interactive activities because they optimize the use of interactive threads.

## Decision Activity

Decision activities allow you to decide the next action to take based on statistics that show you how this situation was solved in the past.

A Decision activity displays information about the instance you are processing and allows you to choose between different actions. Each action provides information about the decision other users took in a similar situation. This information is based on specific process variables.

You should use a Decision activity in those parts of the process where the user has to decide which of the multiple outbound transitions the instance should follow. In this way you can assist the users in making their decisions.

The following is a typical example of the use of a Decision activity: You can classify car insurance policies into low, medium or high risk based on certain variables. The age of the insured, the geographic area and the model of the car are some of the variables that determine the risk. You can model this classification with

a Decision activity. After some time the system learns and suggest the most appropriate answer for each new insurance policy you need to evaluate.

Internally Decision activities use the standard component `Fuego.Bis.DecisionProblem`. You may use this component from PBL methods if you need to implement a complex case that is not covered by the Decision activity. The methods of this component allow you to provide the engine with data to perform the statistical analysis, and to use the results of this analysis. You can use this component from any PBL or BPM Object method.

### **Adding a Decision Activity**

When you add a Decision activity, you should take into account the following considerations.

#### ***Roles***

You must add Decision activities within a role lane in the process. When you try to add a Decision activity outside a role lane, Studio prompts you for an already existing role or allows you to create a new one.

The role lane where you add the Decision activity determines which users can access it. Only the users assigned to that role can run the interactive activity.

#### ***Transitions***

You must connect a Decision activity to the process flow using an inbound and an outbound transition. An error appears in the **Problems View** if the inbound or outbound transitions are missing.

#### ***Variables***

Decision activities can access to the following variables:

- predefined variables
- instance variables

### ***Using Decision Activities as Exception Handlers***

You can use a Decision activity to handle a business exception. The activity that throws a user-defined exception is connected to the Decision activity through an exception transition. For more details about exception handling, see [Exception Handling in Oracle BPM](#) on page 230.

### **Defining the Task of Decision Activity**

You must define the logic of a Decision activity in the main task.

The implementation type of a Decision activity is a task of type Decision. In this task you should specify the instance and predefined variables that the activity shows to the user, and the buttons with the actions user can select.

The user can not edit the instance and predefined variables you select. These variables only show information that helps the user of the process make a decision.

### **Viewing a Decision Activity in WorkSpace**

Describes how WorkSpace displays Decision activities.

You can view and process Decision activities in WorkSpace if you have the required roles.

According to the WorkSpace configuration Decision activities display in one of the following ways:

- A dialog
- A maximized dialog
- A pop-up window
- Within the detail panel

You can search for instances sitting in a Decision activity or define a view that includes them. To view the instance details select the instance in the search result or in the view.

When you run a Decision activity it displays a list of variables that determine which action you should choose. Below those variables it displays the buttons that correspond to the available actions. Each button displays a percentage to guide you making your decision. These percentages show you how similar situations were resolved in the past.

### Running a Decision Activity

Describes how the Process Execution Engine processes a Decision activity.

The Process Execution Engine records your decisions and performs a statistical analysis to be able to provide suggestions for future decisions. It uses an algorithm based on Support Vector Machines (SVMs) methods.

The Decision activity shows you a set of probability percentages. These percentages show you which action users choose, when presented with similar data. Using these percentages you can then make a decision based on the decisions made by previous users.



**Note:** To train the Process Engine you have to provide it a set of consistent data that support the decisions you make. If you make wide fluctuations in the responses during the training period, then the Engine makes inconsistent recommendations.



**Note:** The Process Engine analyses the recorded decisions at periodic intervals as part of the Engine Disposer service. By default the Engine Disposer runs every two days. If you run a decision activity before the Engine Disposer service run, then the Engine does not provide any suggestions.

### Automatic Activity

Automatic activities do not require human involvement. The engine can process Automatic activities on its own.

Automatic activities should use applications and components that do not require human intervention.

You should use Automatic activities for those parts of your process that do not require human intervention.

Typical uses of Automatic activities are:

- Updating databases
- Running batch programs
- Sending e-mail notifications

### Adding an Automatic Activity

When you add an Automatic activity, you should take into account the following considerations.

#### Roles

You should add Automatic activities in automatic role lanes. You can add Automatic activities in user-defined roles but this does not add any information to the process. The engine runs Automatic activities without the user's intervention.

#### Transitions

You must connect an Automatic activity to the process flow using an inbound and an outbound transition. An error appears in the **Problems View** if the inbound or outbound transitions are missing.

#### Variables

The task associated to the Automatic activity can access the following types of variables:

- predefined variables
- instance variables
- local variables

### **Argument Mapping**

You may need to define an argument mapping when you implement an Automatic activity with a component or a procedure task.

When you implement an Automatic activity with a component task, the method you select might require input and output arguments.

In this case you need define the argument mapping for this task to map:

- the instance variables in the process to the input arguments of the component method
- the output arguments of the component method to the instance variables in the process

When you implement an Automatic activity with a procedure task the procedure you select might require might require input and output arguments.

In this case you need define the argument mapping for this task to map:

- the instance variables in the process to the input arguments of the procedure
- the output arguments of the procedure to the instance variables in the process

### **Defining the Task of an Automatic Activity**

You must define the logic of an Automatic activity in the main task.

Automatic activities have only one main task associated to them.

The task of an Automatic activity can have the following implementation types:

- Method
- Component
- Procedure

The components accessed from Automatic activities should not require human involvement. If the component needs input from an end user then you should use an Interactive activity.

You may invoke the display method in the PBL-Method of an Automatic activity for debugging. When you invoke the display method from within an Automatic activity the Process Execution Engine writes the message to the Engine log.

### **Viewing an Automatic Activity in WorkSpace**

Describes how you can view an Automatic activity in WorkSpace.

Automatic activities do not have a user interface because the engine runs them without the involvement of a user. You cannot process Automatic activities.

You can search for instances sitting in an Automatic activity or define a view that includes them. To view the instance details select the instance in the search result or in the view. If the instance is waiting for the engine to process it you can add notes and attachments to it. If the engine is processing the instance adding a note or an attachment causes an error.

You can view information about the processing of an Automatic activity in the audit trail.

### **Running an Automatic Activity**

Describes how the Process Execution Engine processes an Automatic activity.

The Process Execution Engine processes the instances that arrive to an Automatic activity in the order they arrive. If the instance arrives to the Automatic activity when the engine is processing another instance in the same Automatic activity, then the engine queues the instance until it is available to process it.

### **Defining the Task of an Automatic Activity**

You must define the logic of an Automatic activity in the main task.

Automatic activities have only one main task associated to them.

The task of an Automatic activity can have the following implementation types:

- Method



- Component
- Procedure

The components accessed from Automatic activities should not require human involvement. If the component needs input from an end user then you should use an Interactive activity.

You may invoke the display method in the PBL-Method of an Automatic activity for debugging. When you invoke the display method from within an Automatic activity the Process Execution Engine writes the message to the Engine log.

### Handling Errors in an Automatic Activity

Describes how to handle the errors that might occur during the carrying out of an Automatic activity.

You can configure the Process Execution Engine to retry running the activity if it fails on the first attempt to run it.

You can configure the number of times the Process Execution Engine retries to run the activity and the waiting interval between retries. If after trying the specified number of times the Process Execution Engine is not able to complete the activity successfully then it throws an exception.

To revert the changes made by an Automatic activity that did not complete successfully, you should add an exception handling flow to that Automatic activity. You should define an exception flow for all the exception that may arise during the carrying out of the activity. To do this add one exception transition for each exception. If you want to handle unexpected exceptions add an exception transition for the Others exception.

For more information about Exception Handling, see [Handling Exceptions](#) on page 230.

### Subflow Activity

Subflow activities allow you to create an instance in another process.

The process invoked by the Subflow activity is called subprocess. You can use any process as a subprocess.

Use a Subflow activity to:

- **Simplify a process.** You can group related activities in a subprocess and then invoke the subprocess from a Subflow activity. The Subflow activity represents a high level task.
- **Reuse processes.** You can invoke a process from multiple Subflow activities defined in different processes.
- **Enable Business-to-Business (B2B) communication between processes.** A Subflow activity can invoke a process that is running in another Engine that might belong to another company.
- **Distribute work load across multiple Process Executions Engines.** You can distribute the work load of your process by adding Subflow activities that invoke processes that are running in another Process Execution Engine.

### Adding a Subflow Activity

When you add a Subflow activity, you should take into account the following considerations.

After adding a Subflow activity, you must specify the subprocess it invokes.

### Roles

You can add Subflow activities in user-defined or automatic role lanes.

### Transitions

You must connect a Subflow activity to the process flow using an inbound and an outbound transition. An error appears in the **Problems View** if the inbound or outbound transitions are missing.

### Argument Mapping

You may need to define an argument mapping when the subprocess requires input or output arguments.

### Viewing a Subflow Activity in WorkSpace

Describes how you can view a Subflow activity in WorkSpace.

Subflow activities do not have a user interface because the engine runs them without the involvement of a user. You can not process Subflow activities.

You can search for instances sitting in a Subflow activity or define a view that includes them. To view the instance details select the instance in the search result or in the view. If the instance is waiting for the engine to process it you can add notes and attachments to it. If the engine is processing the instance adding a note or an attachment causes an error.

You can view information about the processing of a Subflow activity in the audit trail.

### Running a Subflow Activity

Describes how the Process Execution Engine processes a Subflow activity.

When an instance arrives to a Subflow activity the Process Execution Engine creates an instance in the process associated to the Subflow activity.

If the subprocess requires input arguments, the Engine uses the argument mapping of the Subflow to assign them values.

Then the instance created in the subprocess flows through it.

When the instance reaches the End of the subprocess, the subprocess notifies the Subflow activity. When the Subflow activity receives the notification, it assigns the output arguments of the subprocess to the process instance variables using the defined argument mapping.

### Process Creation Activity

Process Creation activities create an instance in another process and run it asynchronously.

The process invoked by the Process Creation activity is called subprocess. You can use any process as a subprocess.

Use Process Creation activities to:

- **To reduce the time it takes to run a process.** You can simultaneously run more than one process.
- **Simplify a process.** You can group related activities in a subprocess and then invoke the subprocess from a Process Creation activity. The Process Creation activity represents a high level task.
- **Reuse processes.** You can invoke a process from multiple Process Creation activities defined in different processes.
- **Enable Business-to-Business (B2B) communication between processes.** A Process Creation activity can invoke a process that is running in another Engine that might belong to another company.
- **Distribute work load across multiple Process Executions Engines.** You can distribute the work load of your process by adding subflow activities that invoke processes that are running in another Process Execution Engine.

Process Creation activities are similar to Subflow activities because both create an instance in a subprocess.

The main difference with Subflow activities is that Process Creation activities are asynchronous. The instance in the Process Creation activity does not wait for the subprocess to finish, to move to the next activity in the process.

If you need to wait for the subprocess to finish or you need to retrieve its output arguments, you can add a Termination Wait activity later on in the process flow. You can add flow objects between the Process Creation activity and the Termination Wait activity. The Engine runs these flow Objects and the subprocess simultaneously.

### Adding a Process Creation Activity

When you add a Process Creation activity, you should take into account the following considerations.

After adding a subflow activity, you must specify the subprocess it invokes.

By default the Creation Activity run asynchronously and the subprocess does not return arguments to the parent process. If you need to wait for the subprocess to finish or need to retrieve its output arguments, you can add a Termination Wait activity. For more information about Termination Wait activities, see [Termination Wait Activity](#) on page 59.

## Roles

You can add Subflow activities in user-defined or automatic role lanes.

## Transitions

You must connect a Process Creation activity to the process flow using an inbound and an outbound transition. An error appears in the **Problems View** if the inbound or outbound transitions are missing.

## Argument Mapping

You may need to define an argument mapping when the subprocess requires input arguments.

### Viewing a Process Creation Activity in WorkSpace

Describes how you can view a Process Creation activity in WorkSpace.

Process Creation activities do not have a user interface because the engine runs them without the involvement of a user. You can not process Process Creation activities.

You can search for instances sitting in a Process Creation activity or define a view that includes them. To view the instance details select the instance in the search result or in the view. If the instance is waiting for the engine to process it you can add notes and attachments to it. If the engine is processing the instance adding a note or an attachment causes an error.

You can view information about the processing of a Process Creation activity in the audit trail.

### Running a Process Creation Activity

Describes how the Process Execution Engine processes a Process Creation activity.

When an instance arrives to a Process Creation activity, the Process Execution Engine creates an instance in the subprocess associated to the Process Creation activity. If there is no Termination Wait activity associated with the Process Creation activity, the Engine runs the subprocess and the rest of the process flow simultaneously.

If there is a Termination Wait associated to the Process Creation activity, the Engine runs the subprocess and the flow objects that may exist, simultaneously. The instance flows to the next activity when the Engine finishes running the flow objects previous to the Termination Wait. When the instance in the subprocess arrives to the End, the subprocess notifies the associated Termination Wait. Then the Engine maps the output arguments of the subprocess to the process instance variables, using the defined argument mapping.

If the instance in the parent process arrives to the Termination Wait before the Engine finishes running the subprocess, and the property *Process Notification Immediately* is selected, then the Engine processes the notification and the argument mapping in the same transaction.

## Termination Wait Activity

Termination Wait activities add a synchronization point after a Process Creation activity. Adding a Termination Wait activity after a Process Creation activity is optional.

Termination Wait activities are always related to a Process Creation activity. The associated Process Creation must be located previous to the Termination Wait activity in the process flow, and it must have the property *Keep relationship with child process* selected.

For more information about Process Creation activities, see [Process Creation Activity](#) on page 58.

The combination of the Process Creation and its corresponding Termination Wait activity is very similar to a Subflow activity. The advantage of using a Process Creation and a Termination Wait is that you can add multiple flow objects between them. The Process Engine executes this flow object and the subprocess associated to the Process Creation Activity simultaneously.

### Adding a Termination Wait Activity

When you add a Termination Wait activity, you should take into account the following considerations.

After adding a Termination Wait activity you must associate it with a Process Creation activity. The Process Creation activity must be located previous to the Termination Wait activity in the process flow, and must have the *Keep relation with child process* property selected.

For more information about Process Creation activities, see [Process Creation Activity](#) on page 58.

### Roles

You should add Termination Wait activities in automatic role lanes. You can add Termination Wait activities in user-defined roles but this does not add any information to the process. The engine runs Termination Wait activities without the user's intervention.

### Transitions

You must connect a Termination Wait activity to the process flow using an inbound and an outbound transition. An error appears in the **Problems View** if the inbound or outbound transitions are missing.

### Argument Mapping

You may need to define an argument mapping when the subprocess has output arguments.

### Viewing a Termination Wait Activity in WorkSpace

Describes how you can view a Termination Wait activity in WorkSpace.

Termination Wait activities do not have a user interface because the engine runs them without the involvement of a user. You can not process Termination Wait activities

You can search for instances sitting in a Termination Wait activity or define a view that includes them. To view the instance details select the instance in the search result or in the view.

You can view information about the processing of a Termination Wait activity in the audit trail.

### Running a Termination Wait Activity

Describes how the Process Execution Engine processes a Termination Wait activity.

The instance flows to a Termination Wait activity after the Engine finishes running the flow objects that might exist between the Process Creation activity and the Termination Wait Activity. When the instance in the subprocess arrives to the End, the subprocess notifies the associated Termination Wait. Then the Engine maps the output arguments of the subprocess to the process instance variables, using the defined argument mapping.

If the instance in the parent process arrives to the Termination Wait activity before the Engine finishes running the subprocess, and the property *Process Notification Immediately* is selected, then the Engine processes the notification and the argument mapping in the same transaction.

### Grab Activity

Grab activities allow you to move an instance from one activity to another, or to reassign it to another user.

You can use a grab activity to allow users in supervisory roles to control the instance flow. If necessary they can run this activity to move an instance to another activity or to assign it to another user.

Grab activities give processes the flexibility to deal with slowdown conditions and to redistribute instances to alleviate such conditions.

Typically you provide access to Grab activities only to supervisory roles.

### Adding a Grab Activity

When you add a Grab activity, you should take into account the following considerations.

After you add a Grab you need to select a Grab type. The Grab type defines the transitions that the Grab activity allows.

## Roles

You must add Grab activities within a role lane in the process. When you try to add a grab activity outside a role lane, Studio prompts you for an already existing role or allows you to create a new one.

The role lane where you add the Grab activity determines which users can access it. Only the users assigned to that role can run the Grab activity.

## Variables

The task associated to the Interactive activity can access the following types of variables:

- predefined variables
- instance variables
- arguments
- local variables

## Transitions

- **Defined Grab:** You must add inbound and outbound transitions to indicate from which flow objects you can grab the instance and to which flow objects you can redirect it.

If a Defined Grab activity has transitions to or from flow objects in a Split-Join circuit, then it cannot have transitions to or from flow objects outside the Split-Join circuit. Similarly, a Grab activity with transitions to or from flow objects outside a Split-Join circuit cannot have transitions to or from flow objects located in a Split-Join circuit.

- **Grab From All:** The Grab From All activity can grab instances from any flow object in the process. You cannot add an inbound transition to a Grab From All Activity because it implicitly has inbound transitions from all the flow objects in the process. You must add outbound transitions to indicate to which flow objects you can redirect the grabbed instance.
- **Grab From All/To All:** The Grab From All/To All activity implicitly has an inbound and an outbound transition to all the flow objects in the process. Thus you cannot add any transitions to this type of Grab activity.

## Defining the Task of a Grab Activity

You might need to define a main task to allow the user running the Grab activity to make changes to the instance data. If appropriate you might define optional tasks to help the user run the main task.

There are different options to implement the tasks of a Grab activity. Each option addresses specific types of interaction.

The tasks of a Grab activity can have the following implementation types:

- Method
- Component
- Screenflow
- External
- Input
- Display

The default implementation type is method.

For more information about the different implementation types, see [Flow Object Tasks](#) on page 101.

## Optional Tasks

You can run optional tasks while you are running the main task of a Grab activity. These tasks are usually related to the main task. They provide information that you might need to complete the Grab activity.

Optional tasks are repeatable and read-only. They are not allowed to modify the instance information because this information is used by the main task.

### **Viewing a Grab Activity in WorkSpace**

Describes how WorkSpace displays Grab activities.

You can view and process Grab activities in WorkSpace if you have the required role. If the instance is sitting in an activity that has an implicit or explicit transition to a Grab activity, an icon appears next to it to indicate that you can grab this instance.

If the Grab activity has a main task defined then WorkSpace shows the main task after you click on the icon to grab the instance. According to the WorkSpace configuration, the tasks of a Grab activity displays in one of the following ways:

- A dialog
- A maximized dialog
- A pop-up window
- Within the detail panel

After you complete the main task the instance flows to the next flow object according to the explicit or implicit transitions. If the Grab activity is of the type From All/To All, it allows the user to select where it should flow.

You can search for instances sitting in a Grab activity or define a view that includes them. To view the instance details select the instance in the search result or in the view.

You cannot grab an instance that is sitting in another Grab activity.

### **Running a Grab Activity**

Describes how the Process Execution Engine processes a Grab activity.

Generally Grab activities require human involvement to complete their task. Only after the user completes the Grab task associated to the activity, the instance moves to the next flow object in the process flow. If the Grab activity is of type From All/To All then the user must select the next flow object.

The Engine processes Grab activities using interactive threads. If the implementation of the Grab activity is of type screenflow, then the engine releases the interactive thread at the beginning of the screenflow and obtains a new interactive thread when the user completes the screenflow. When possible you should use screenflows to implement Grab activities because they optimize the use of interactive threads.

## **Gateways**

This section provides information about the gateways you can use to design a process.

### **Conditional Gateway**

Conditional gateways allow you to model conditional process flows.

Conditional gateways make the process easier to read. There is no difference between using conditional transitions with or without a Conditional gateway.

### **Adding a Conditional Gateway**

When you add a Conditional gateway, you should take into account the following considerations.

#### ***Roles***

You should add Conditional gateways in automatic role lanes. You can add Conditional gateways in user-defined roles but this does not add any information to the process. The engine runs Conditional gateways without the user's intervention.

#### ***Transitions***

You must connect a Conditional gateway to the process flow using an inbound and an outbound transition. An error appears in the **Problems View** if the inbound or outbound transitions are missing.

Generally Conditional gateways have multiple outbound conditional transitions. You can decide the order to evaluate these transitions.

### **Viewing a Conditional Gateway in WorkSpace**

Describes how you can view a Conditional gateway in WorkSpace.

Conditional gateways do not have a user interface because the engine runs them without the involvement of a user. You can not process Conditional gateways.

You can search for instances sitting in a Conditional gateway or define a view that includes them. To view the instance details select the instance in the search result or in the view. If the instance is waiting for the engine to process it you can add notes and attachments to it. If the engine is processing the instance adding a note or an attachment causes an error.

You can view information about the processing of a Conditional gateway in the audit trail.

### **Running a Conditional Gateway**

Describes how the Process Execution Engine processes a Conditional Gateway.

The Process Execution Engine processes the instances that arrive to a Conditional Gateway in the order they arrive. If the instance arrives to the Conditional Gateway when the engine is processing another instance in the same Conditional Gateway, then the engine queues the instance until it is available to process it.

The Process Execution Engine evaluates the outbound conditional transitions in the order you defined in the Conditional gateway.

### **Split Gateway**

A Split activity allows an instance to simultaneously run through multiple process paths.

The number of copies that the Split gateway generates corresponds to the number of outbound unconditional transitions plus any outbound conditional transitions that evaluate to True.

Split activities must have a corresponding Join activity in order to complete the circuit and resume the process flow.

### **Adding a Split Gateway**

When you add a Split gateway, you should take into account the following considerations.

When you add a Split gateway, Studio adds the corresponding Join gateway. After adding the Split gateway you need to define the different threads by adding flow objects and transitions between the Split and the Join gateway.

You can configure the Split-Join circuit to use multiple copies of the process instance, or to use the same process instance for all the defined threads. To do this you need to edit the Split properties and modify the Generate Copies property.

Typically you select the Generate Copies properties when the different threads do not modify the instance information. If any of the threads modify the instance information you need to define how to merge this information by adding PBL code to the Join gateway. If you do not define how to merge the instance data then the modifications are lost.

If you need to share information between the different threads of the Split-Join circuit then you should not select Generate Copies. In this way all the threads use and modify the original instance.

### **Roles**

You should add Split-Join circuits in automatic role lanes. You can add Split-Join circuits in user-defined roles but this does not add any information to the process. The engine runs the Split and Join gateways without the user's intervention.

## Transitions

You must connect the Split gateway to the process flow using an inbound transition. You must add one outbound unconditional transition to the Split gateway. You may add multiple outbound conditional and unconditional transitions.

The Join gateway in a Split-Join circuit allows multiple inbound transitions. You must connect the Join gateway to the process flow using one or more outbound transitions.

An error appears in the **Problems View** if the inbound or outbound transitions that connect the Split-Join circuit to the process flow are missing.

The Split gateway must have at least one outbound unconditional transition. It may have one or more outbound conditional and unconditional transitions.

The flow objects within a Split-Join circuit can have transitions to other flow object within the Split-Join circuit. You cannot define transitions to flow objects outside the Split-Join circuit, with the exception of Grab activities. However the outbound transition of the Grab activity should connect it to an activity in the Split-Join circuit or to the End of the process.

## Variables

The PBL-Methods associated to the Split and the Join gateway can access the following types of variables:

- predefined variables
- instance variables
- local variables
- arguments

## Viewing a Split Gateway in WorkSpace

Describes how you can view a Split gateway in WorkSpace.

Split and Join gateways do not have a user interface because the engine runs them without the involvement of a user. You cannot process Split and Join gateways.

You can search for instances sitting in a Split or a Join gateway, or define a view that includes them. To view the instance details select the instance in the search result or in the view. If the instance is waiting for the engine to process it you can add notes and attachments to it. If the engine is processing the instance adding a note or an attachment causes an error.

You can view information about the processing of a Split or a Join gateway, in the audit trail.

## Running a Split Gateway

Describes how the Process Execution Engine processes Split gateway.

When an instance reaches a Split gateway, the original instance immediately flows to the corresponding Join gateway.

If the Generate Independent Copies property of the Split gateway is selected, the Split gateway creates a copy of the instance for each of the threads of the Split-Join circuit. If any of the threads modifies the instance data of the copies the other copies do not have access to the modified data. When the instance arrives to the Join gateway the Engine runs the associated PBL-Method, generally this method merges the data in the different copies and copies it to the original instance.



**Note:** The creation time of the instance copy corresponds to the time when the Process Engine created the copy. This value is stored in the variable `time` of the predefined variable `creation`.

If the Generate Independent Copies property of the Split gateway is not selected, the Engine tries to run the threads simultaneously. If one of the threads modifies the instance data the Engine locks the instance, so if another thread needs to access the instance data it has to wait until the Engine unlocks it. When the instance arrives to the Join gateway the Engine runs the associated PBL-Method. Because all the threads use the original instance it is not necessary to merge data in the Join gateway.



## Or-Split Gateway

You can use an Or-Split gateway to model alternative but non-exclusive paths and specify how to proceed if none of these paths are feasible.

It represents a branching point in the process, where the process flow divides into alternative paths.

The Or-Split gateway provides a subset of the Split functionality. It contemplates multiple possibilities and clearly states how to proceed if none of those possibilities are valid.

### Adding an Or-Split Gateway

When you add an Or-Split gateway, you should take into account the following considerations.

When you add an Or-Split gateway, Studio adds the corresponding Join gateway. After adding the Or-Split gateway you need to define the different threads by adding flow objects and transitions between the Or-Split and the Join gateway.

You can configure the Or-Split-Join circuit to use multiple copies of the process instance, or to use the same process instance for all the defined threads. To do this you need to edit the Or-Split properties and modify the Generate Copies property.

Typically you select the Generate Copies properties when the different threads do not modify the instance information. If any of the threads modify the instance information you need to define how to merge this information by adding PBL code to the Join gateway. If you do not define how to merge the instance data then the modifications are lost.

If you need to share information between the different threads of the Or-Split-Join circuit then you should not select Generate Copies. In this way all the threads use and modify the original instance.

## Roles

You should add Or-Split-Join circuits in automatic role lanes. You can add Or-Split-Join circuits in user-defined roles but this does not add any information to the process. The engine runs the Or-Split and Join gateways without the user's intervention.

## Transitions

You must connect the Or-Split gateway to the process flow using an inbound transition. You must add one outbound unconditional transition to the Or-Split gateway. You may add it one or more outbound conditional transitions.

The Join gateway in an Or-Split-Join circuit allows multiple inbound transitions. You must connect the Join gateway to the process flow using one or more outbound transitions.

An error appears in the **Problems View** if the inbound or outbound transitions that connect the Or-Split-Join circuit to the process flow are missing.

The flow objects within an Or-Split-Join circuit can have transitions to other flow object within the Or-Split-Join circuit. You cannot define transitions to flow objects outside the Or-Split-Join circuit, with the exception of Grab activities. However the outbound transition of the Grab activity should connect it to an activity in the Or-Split-Join circuit or to the End of the process.

## Variables

The PBL-Methods associated to the Or-Split and the Join gateway can access the following types of variables:

- predefined variables
- instance variables
- local variables
- arguments

## Viewing an Or-Split Gateway in Workspace

Describes how you can view an Or-Split gateway in Workspace.

Or-Split and Join gateways do not have a user interface because the engine runs them without the involvement of a user. You cannot process Or-Split and Join gateways.

You can search for instances sitting in an Or-Split or a Join gateway, or define a view that includes them. To view the instance details select the instance in the search result or in the view. If the instance is waiting for the engine to process it you can add notes and attachments to it. If the engine is processing the instance adding a note or an attachment causes an error.

You can view information about the processing of an Or-Split or a Join gateway, in the audit trail.

### **Running an Or-Split Gateway**

Describes how the Process Execution Engine processes Split gateway.

When an instance reaches an Or-Split gateway, the original instance immediately flows to the corresponding Join gateway.

Then the Engine evaluates the Or-Split conditional transitions and for each transitions that evaluates to true. If none of the conditional transitions evaluated to true, then the Engine creates a thread for the unconditional transition.

If the Generate Independent Copies property of the Or-Split gateway is selected, the Or-Split gateway creates a copy of the instance for each of the threads of the Or-Split-Join circuit. If any of the threads modifies the instance data of the copies the other copies do not have access to the modified data. When the instance arrives to the Join gateway the Engine runs the associated PBL-Method, generally this method merges the data in the different copies and copies it to the original instance.

If the Generate Independent Copies property of the Or-Split gateway is not selected, the Engine tries to run the threads simultaneously. If one of the threads modifies the instance data the Engine locks the instance, so if another threads needs to access the instance data it has to wait until the Engine unlocks it. When the instance arrives to the Join gateway the Engine runs the associated PBL-Method. Because all the threads use the original instance it is not necessary to merge data in the Join gateway.

### **Multiple Gateway**

Multiple gateways allow you to create multiple copies of an instance so that different participants can process the same instance simultaneously.

You should use a Multiple gateway if you need to process multiple copies of the same instance simultaneously. All the instance copies follow the same process flow. Generally a different user processes each of the copies.

A use case example of a Multiple Gateway is a process that solicits bids from external vendors. The company wants to get multiple bids from different vendors and select the lowest bid that meets the company's specifications.

### **Adding a Multiple Gateway**

When you add a Multiple gateway, you should take into account the following considerations.

When you add a Multiple gateway, Studio automatically adds the corresponding Join gateway, and the PBL-Method associated to the Multiple gateway.

When you edit the Join gateway Studio adds the corresponding PBL-Method, and associates it to the Join gateway.

In a Multiple-Join circuit you must create the instance copies using PBL code.



**Note:** The Interactive activities you add within the Multiple-Join circuit must not be Suspendable.

### **Roles**

You should add Multiple gateways in automatic role lanes. You can add Multiple gateways in user-defined roles but this does not add any information to the process. The engine runs Multiple gateways without the user's intervention.

## Variables

The PBL-Methods associated to the Multiple and the Join gateway can access the following types of variables:

- predefined variables
- instance variables
- local variables
- arguments

## Transitions

You must connect the Multiple gateway to the process flow using an inbound transition. You can add only one outbound transition to the Multiple gateway.

The Join gateway in a Multiple-Join circuit allows only one inbound transition. You must connect the Join gateway to the process flow using one or more outbound transitions.

An error appears in the **Problems View** if the inbound or outbound transitions that connect the Multiple-Join circuit to the process flow are missing.

The flow objects within a Multiple-Join circuit can have transitions to other flow object within the Split-Join circuit. You cannot define transitions to flow objects outside the Multiple-Join circuit, with the exception of Grab activities. However the outbound transition of the Grab activity should connect it to an activity in the Multiple-Join circuit or to the End of the process.

## Creating the Instance Copies

To generate instance copies you must add the following code to the PBL-Method associated to the Multiple gateway:

```
i = 0
while i < numberOfCopies
do
    // Create a copy of the process instance.
    copy= clone(this)

    // Get ready for next loop.
    i = i + 1
end
```

**Join** activities can also access instance copies by using **copy**. as shown below:

To access the instance copies from the Join gateway you must add the following code to the associated PBL-Method:

```
//Assign the instance variable the value of the variable in
the copy.
variableName = copy.variableName
```

## Viewing a Multiple Gateway

Describes how you can view a Multiple gateway in Workspace.

Multiple and Join gateways do not have a user interface because the engine runs them without the involvement of a user. You cannot process Multiple and Join gateways.

You can search for instances sitting in a Multiple or a Join gateway, or define a view that includes them. To view the instance details select the instance in the search result or in the view. If the instance is waiting for the engine to process it you can add notes and attachments to it. If the engine is processing the instance adding a note or an attachment causes an error.

You can view information about the processing of a Multiple or a Join gateway, in the audit trail.

### Running a Multiple Gateway

Describes how the Process Execution Engine processes Split gateway.

When the instance arrives to the Multiple gateway the Process Execution Engine runs the PBL-Method associated to the Multiple gateway. The PBL-Method creates copies of the original instance.

The Engine processes the multiple instance copies simultaneously using the flow defined between the Multiple gateway and the Join gateway.

The original instance flows to the Join gateway and stays there until:

- all the instance copies arrive to the Join gateway.
- the number of copies specified by the property Number of copies to wait to release, arrive to the Join gateway.
- the PBL-Method associated to the Join gateway sets the value of the predefined variable action to RELEASE.
- the process reaches it deadline.
- a due transition that connects the Join to the process flow expires.

If you add or modify the attachments of an instance copy, the Engine automatically updates the attachments in the original instance when it reaches the Join gateway.



**Note:** If you modify the values of the instance variables in the copies, then the original instance uses the values of the last copy that arrives to the Join gateway. If you want to select or merge the values of the variables in the instance copies you can then you must add the PBL code to do this in the PBL-Method associated to the Join gateway.

## Events

This section provides information about the events you can use to design a process.

### Begin Event

The Begin event provides an entry point to the process.

The Begin event creates an instance in the process and assigns values to the process instance variables.

There is only one Begin event per process.

You can trigger a Begin event running:

- a Global Creation activity.
- a Global Automatic activity.
- a Subflow activity located in a parent process.
- a Process Creation activity in another process.
- an external application or web page that uses PAPI or WAPI.

### Working with a Begin Event

When you work with a Begin event, you should take into account the following considerations.

When you create a process, Studio automatically adds a Begin and an End event connected by an unconditional transition.

## ***Roles***

Studio automatically adds a Begin event in an automatic role lane when you create a new process. You can move a Begin event to a user-defined role but this does not add any information to the process. The engine runs Begin events without the user's intervention.

## ***Transitions***

You cannot add an inbound transition to a Begin event. You must connect the Begin event to the process flow through one unconditional outbound transition. You may add it multiple outbound conditional transitions.

If you define multiple Argument Sets, then Message based transitions become available.

## ***Variables***

A Begin event can access the following types of variables:

- predefined variables
- instance variables
- arguments
- local variables

## ***Argument Mapping***

You can add input arguments to a process using the Argument Mapping of the Begin event. In the Argument Mapping you define how to map the input arguments to the process instance variables.

The input arguments can come from:

- a Global Creation activity.
- a Process Creation activity.
- a Subflow activity.
- a Global Automatic activity running PBL code to create instances.
- a Global activity running PBL code to create instances.

Advance scripting is available for compatibility with previous versions. Oracle recommends not to use this feature. If you need to run code immediately after the instance creation you can add an automatic activity immediately after the Begin event.

## **Viewing a Begin Event in WorkSpace**

Describes how you can view a Begin event in WorkSpace.

Begin events do not have a user interface because the engine runs them without the involvement of a user. You cannot process Begin events.

You can view information about the processing of a Begin event in the audit trail.

## **End Event**

The End event is the last activity in any process. It is the exit point of the process.

There is only one end per process. Studio automatically adds a Begin and an End event when you create a process.

If you use the process as a subprocess the End event notifies and returns information to the Process Creation or Subflow activities in the parent process.

## **Working with an End Event**

When you work with an End event, you should take into account the following considerations.

When you create a process, Studio automatically adds a Begin and an End event connected by an unconditional transition.

**Roles**

Studio automatically adds an End event in an automatic role lane when you create a new process. You can move an End event to a user-defined role but this does not add any information to the process. The engine runs End events without the user's intervention.

**Transitions**

You must connect the process flow to the End event using at least one inbound transition. You cannot add an outbound transition to an End event.

You can grab instances sitting in an End event if they are not aborted or terminated.

**Variables**

A Begin event can access the following types of variables:

- predefined variables
- instance variables
- arguments
- local variables

**Argument Mapping**

If you use the process as a subprocess you can define an Argument Mapping with output arguments to return information to the parent process.

Advance scripting is available for compatibility with previous versions. Oracle recommends not to use this feature. If you need to run code immediately after the instance creation you can add an automatic activity immediately after the Begin event.

**Viewing an End Event in WorkSpace**

Describes how you can view an End event in WorkSpace.

End events do not have a user interface because the engine runs them without the involvement of a user. You cannot process End events.

You can view information about the processing of an End event in the audit trail.

**Message Wait Event**

The Message Wait Activity halts an instance until it receives a notification from another process or an external application.

Message Wait events halt the instance until they receive a notification or until any outbound due transition expires.

Message Wait events receive notifications from:

- a parent process.
- a child process.
- an external application.

**Message Wait Event Types**

The following table describes the different types of events that trigger the Message Wait event.

Event Type	Description
Parent Process	The Message Wait event waits for a notification from a Send Message event in the parent process.

Event Type	Description
Child Process	<p>To use this option you must select the property Keep relation with child in the corresponding Process Creation activity in the parent process.</p> <p>You must associate the Process Creation activity that created the child instance, to the Send Message event using the Related Process Property in the Send Message event. And you must place the Send Message event between the Process Creation activity and the Termination Wait activity in the parent process.</p>
	<p>The Message Wait event waits for a notification from a Send Message event in the child process.</p> <p>To use this option you must select the property Keep relation with child in the corresponding Process Creation activity in the parent process.</p> <p>You must associate the Process Creation activity that created the child instance, to the Message Wait event using the Related Process Property in the Message Wait event. And you must place the Message Wait activity between the Process Creation activity and the Termination Wait activity in the parent process.</p>
External Event	<p>The Message Wait event waits for a notification from an external application. Possible external applications are:</p> <ul style="list-style-type: none"> <li>• A PAPI application</li> <li>• An activity in an external process that uses the <code>send()</code> method from the <code>Notification</code> component</li> <li>• A process that is not a subprocess of the current process and contains a Send Message event that notifies the current process</li> </ul>

### Interruptions

Using a Message Wait event with the property Allow Interruptions enabled, allows you to pull out instances from the process flow when a certain event occurs. The Message Wait Event flow defines how to process the affected instances in reaction to the occurred event.

A Message Wait activity with the property Allow Interruptions enabled behaves like a listener that waits for a notification to pull out instances from the process flow and process them.

It is not part of the process flow. It defines a flow of its own. When a certain event occurs the instances in the process flow are redirected to the Message Wait event flow.

Generally the notification received by the Message Wait event states which instances should be pulled out from the process flow.

The last flow object in the Message Wait flow must return the instance back to the main process flow. You can do this by setting the value of the predefined variable `action` to `BACK`.

### Adding a Message Wait Event

When you add a Message Event event, you should take into account the following considerations.

## **Roles**

You should add Message Wait events in automatic role lanes. You can add Message Wait events in user-defined roles but this does not add any information to the process. The engine runs Message Wait events without the user's intervention.

## **Transitions**

You must connect a Message Wait event to the process flow using an inbound and an outbound transition. An error appears in the **Problems View** if the inbound or outbound transitions are missing.

You can add an outbound due transition to allow instances to move to the next activity if it does not receive a notification within a specified time period.

If you define multiple Argument Sets, then Message based transitions become available.

If you configure the Message Wait event to allow interruptions then you must not connect it to the process flow. The Message Wait event defines a separate flow.

## **Argument Mapping**

You can define an Argument Mapping for the Message Wait event to receive information from the process or the application that triggers the Message Wait event.

### **Viewing a Message Wait Event in WorkSpace**

Describes how you can view a Message Wait event in WorkSpace.

Message Wait events do not have a user interface because the engine runs them without the involvement of a user. You cannot process Message Wait events.

You can search for instances sitting in a Message Wait event or define a view that includes them. To view the instance details select the instance in the search result or in the view. If the instance is waiting for the engine to process it you can add notes and attachments to it. If the engine is processing the instance adding a note or an attachment causes an error.

You can view information about the processing of a Message Wait event in the audit trail.

### **Running a Message Wait Event**

Describes how the Process Execution Engine processes a Message Wait event.

When an instance arrives to a Message Wait event it waits there until the specified event occurs. If the Message Wait event is connected to the process flow by a due transition, the instance leaves the Message Wait activity when the due transition expires.

When the Message Wait event receives a notification from another process or application the Process Execution Engine performs the Argument mapping and then moves the instance to the next activity in the process flow.

If the Message Wait has the property Allow Interruptions enabled, the Message Wait is not part of the process flow. The Engine is constantly monitoring for an event to occur. When this event occurs the Engine pulls out from the process flow the instances that match the criteria specified in the notification. Then the instances flow through the Message Wait flow and once they get to the last activity they are sent back to the main process flow.

### **Send Message Event**

Send Message events allow you to notify an instance in another process.

You can combine Send Message and Message Wait events to allow communication between processes.

When an instance arrives to a Send Message event the Engine notifies the associated Message Wait activity. Then the Message Wait activity releases the instance, allowing it to move to the next activity.

You can use a Send Message event to notify an instance in:

- a parent process.
- a child process.



- an external process.

You can only notify already existing instances. If you try to notify an instance that does not exist, the Engine throws an Exception.

For more information about Message Wait events, see [Message Wait Event](#) on page 70.

### **Adding a Send Message Event**

When you add a Send Message event, you should take into account the following considerations.

#### **Roles**

You should add Send Message events in automatic role lanes. You can add Send Message events in user-defined roles but this does not add any information to the process. The engine runs Send Message events without the user's intervention.

#### **Transitions**

You must connect the Send Message event to the process flow using an inbound transition. You may add multiple inbound and outbound transitions if necessary.

#### **Argument Mapping**

You can define an Argument Mapping for the Send Message event to send information to the process it notifies.

#### **Viewing a Send Message Event in WorkSpace**

Describes how you can view a Send Message event in WorkSpace.

Send Message events do not have a user interface because the engine runs them without the involvement of a user. You cannot process Send Message events.

You can search for instances sitting in a Send Message event or define a view that includes them. To view the instance details select the instance in the search result or in the view. If the instance is waiting for the engine to process it you can add notes and attachments to it. If the engine is processing the instance adding a note or an attachment causes an error.

You can view information about the processing of a Send Message event in the audit trail.

#### **Running a Send Message Event**

Describes how the Process Execution Engine processes a Send Message event.

When an instance reaches a Send Message event the Engine immediately sends a notification to the corresponding Message Wait activity.

If the Engine succeeds delivering the notification the instance moves to the next activity in the process flow.

#### **Timer Event**

Timer event allow you to delay an instance for an specified time interval or until a certain date.

You can configure a Timer event to run:

- periodically, using a fixed interval.
- daily, weekly or monthly at a specified time.
- periodically, using an interval expressed in PBL code.

#### **Adding a Timer Event**

When you add a Timer event, you should take into account the following considerations.

#### **Roles**

You should add Timer events in automatic role lanes. You can add Timer events in user-defined roles but this does not add any information to the process. The engine runs Timer events without the user's intervention.

## **Transitions**

You must connect a Timer event to the process flow using an inbound and an outbound transition. An error appears in the **Problems View** if the inbound or outbound transitions are missing.

## **Variables**

The PBL code associated to a Timer event can access the following types of variables:

- predefined variables
- instance variables
- local variables

## **Viewing a Timer Event in WorkSpace**

Describes how you can view a Timer event in WorkSpace.

Timer events do not have a user interface because the engine runs them without the involvement of a user. You cannot process Timer events.

You can search for instances waiting in a Timer event or define a view that includes them. To view the instance details select the instance in the search result or in the view. If the instance is waiting for the engine to process it you can add notes and attachments to it. If the engine is processing the instance adding a note or an attachment causes an error.

You can view information about the processing of a Timer event in the audit trail.

## **Running a Timer Event**

Describes how the Process Execution Engine processes a Timer event.

When an instance arrives to a Timer event it waits there until the Engine runs the Timer Event.

The Engine runs the Timer event periodically using the interval you specified in the Timer Event properties.

## **Compensate Event**

Compensate activities allow you to revert the changes made by activities included in a compensation or an exception flow.

Use a Compensate event to compensate all the activities in a compensation flow. You can only add Compensation events to compensation flows or exception flows. Generally you Compensate events to an exception flow or to a group that contains an exception flow.

If you do not specify the activity to compensate the Engine uses the default compensation mechanism of the group. The default compensation order of a group is the inverted execution order.

If you specify an activity to compensate the Engine invokes the compensation of this activity.

The default behaviour of a group when an internal exception occurs within is to automatically compensates all the activities that were run successfully.

## **Adding a Compensate Event**

When you add a Compensate event, you should take into account the following considerations.

The following restrictions apply to Compensate events:

- You can add Compensate events only to exception or compensation flows.
- The Compensate event can only reference flow objects included in the group affected by the compensation flow.
- You can add a Compensate event to a compensation flow or an exception flow that does not contain a group, although it does not add any functionality because there are not any internal executions to compensate.

**Roles**

You should add Compensate events in automatic role lanes. You can add Compensate events in user-defined roles but this does not add any information to the process. The engine runs Compensate events without the user's intervention.

**Transitions**

You must connect Compensate event to the compensation flow using an inbound compensate transition.

**Viewing a Compensate Event in Workspace**

Describes how you can view a Compensate event in Workspace.

Compensate events do not have a user interface because the engine runs them without the involvement of a user. You cannot process Compensate events.

You can view information about the processing of an Automatic activity in the audit trail. The Compensation Event appears multiple times in the Audit Trail because the Process Execution Engine executes a Compensation event multiple times until the compensate flow is compensated.

**Running a Compensate Event**

Describes how the Process Execution Engine processes a Compensate activity.

When an instance arrive to a Compensate event the Engine immediately read the compensation log to determine to which Object Flow it should delegate the control to compensate.

When it finishes compensating this activity the Engine looks for the next activity to compensate.

**Global Activities**

This section provides information about the global activities you can use to design a process.

**Global Creation Activity**

Global Creation activities allow you to create new instances in a process.

Use Global Creations activities to enable certain users to create instances in the process.

You can configure a Global Creation activity to receive input from the user and use this input to create an instance in the process.

**Adding a Global Creation Activity**

When you add a Global Creation activity, you should take into account the following considerations.

**Roles**

You must add Interactive activities within a role lane in the process. When you try to add an Interactive activity outside a role lane, Studio prompts you for an already existing role or allows you to create a new one.

The role lane where you add the Interactive activity determines which users can access it. Only the users assigned to that role can run the Interactive activity.

**Variables**

The task associated to a Global Creation activity can access the following types of variables:

- predefined variables that do not require an instance
- arguments
- local variables

**Transitions**

There are no transitions to or from the Global Creation activity. There is an implied transition from the Global Creation to the Begin activity.

### Defining the Task of a Global Creation

You must define the logic of a Global Creation activity in the main task.

There are different options to implement the tasks of a Global Creation activity.

The tasks of a Global Creation activity can have the following implementation types:

- method
- screenflow

The default implementation type is method.

For more information about the different implementation types, see [Flow Object Tasks](#) on page 101.

### Viewing a Global Creation in WorkSpace

Describes how WorkSpace displays Global Creation activities.

You can view and process Global Creation activities in WorkSpace if you have the required role. Global Creation activities appear in the Applications panel. To run a Global Creation activity click on the link in the Applications panel.

According to the WorkSpace configuration the main task of the Global Creation activity displays in one of the following ways:

- A dialog
- A maximized dialog
- A pop-up window

You can view information about the processing of a Global Creation activity in the audit trail.

### Running a Global Creation

Describes how the Process Execution Engine processes a Global Creation activity.

If the implementation type of the main task of the Global Creation activity is of type Screenflow then the user must process this activity before the Process Execution Engine creates an instance in the process.

If the implementation type is of type Method then it depends on the components this method invokes.

After the user processes the Global Creation activity the Engine creates a new instance in the process.

### Global Automatic

Global Automatic activities allow you to periodically run a component or an application.

Global Automatic activities do not have any direct end user interaction. Typically they invoke applications or components that run on a remote server.

You can use Global Automatic activities to:

- process batch reports.
- download a set of files at a schedules time.
- listen for an specific event in the process.
- listen to a port.
- check for specific events, such as a mouse-click or a broken connection error in a remote component.
- invoke a component or application to create new process instances.

Global Automatic activities are not connected to the process flow. However the events that occur in the process flow may trigger its execution.

### Adding a Global Automatic

When you add a Global Automatic activity, you should take into account the following considerations.

When you add a Global Automatic activity Studio creates a PBL-Method and associates it to the Global Automatic activity. This PBL-Method contains the logic of the activity.

If the type of the Global Automatic activity is Execute when an event occurs, then Studio creates two PBL-Methods. One method has the same name of the activity and the remaining one has the suffix "\_Listening".

The Engine runs the first method when it starts-up or when you deploy the process. This method creates the listening component. When the event occurs it triggers the method with the "\_Listening" suffix.

### ***Roles***

You should add Automatic activities in automatic role lanes. You can add Automatic activities in user-defined roles but this does not add any information to the process. The engine runs Automatic activities without the user's intervention.

### ***Variables***

The PBL-Method associated to Global Automatic activity can access predefined and local variables.

### ***Transitions***

There are no transitions to or from the Global Automatic activity.

### **Viewing a Global Automatic in WorkSpace**

Describes how you can view a Global Automatic activity in WorkSpace.

Global Automatic activities do not have a user interface because the engine runs them without the involvement of a user. You can not process Global Automatic activities

### **Running a Global Automatic**

Describes how the Process Execution Engine processes a Global Automatic activity.

The way the Process Execution Engine runs a Global Automatic activity depends on its type.

### ***Polling by Interval***

The Process Execution Engine waits for the specified time interval to run the PBL-Method. When the Engine finishes running the PBL-Method it starts waiting for the specified time interval to run the PBL-Method again.

### ***Executes when an Event Occurs (event listener)***

When you start the Process Execution Engine or deploy a process, the Engine run the method that creates the listening component.

Then Engine waits for a specified event to occur to run the Listening PBL Method.

### **Automatic Scheduled**

The Process Execution Engine runs the PBL-Method at the specified times.

### **Automatic JMS Listener**

The Process Execution Engine waits for JMS message to run the PBL-Method.

This type is applicable for J2EE Enterprise Editions. PBL-Method runs when a predefined event occurs. The PBL-Method requires argument a Fuego.Msg.JmsMessage that is a wrapper of the javax.jms.Message.

### **Automatic JMX Listener**

The Process Execution Engine waits for JMX message to run the PBL-Method.

### **Global Interactive Activity**

Global Interactive activities allow you to run applications or database queries, that are not directly related on an instance in the process.

You can use Global Interactive activities to:

- run applications that provide the user contextual information to run the process.

- send e-mails.
- retrieve information from a database.

You can configure a Global Interactive activity to run the following predefined applications:

- **Process image:** shows the location of instance within the process graphic.
- **Display instance:** customizes the instance panel.
- **Workload:** shows the workload each of the flow objects in the process.
- **Dashboard:** displays a dashboard with Business Activity Monitoring (BAM) information.

### Adding a Global Interactive Activity

When you add a Global Interactive activity, you should take into account the following considerations.

#### **Roles**

You must add Global Interactive activities within a role lane in the process. When you try to add a Global Interactive activity outside a role lane, Studio prompts you for an already existing role or allows you to create a new one.

The role lane where you add the Global Interactive activity determines which users can access it. Only the users assigned to that role can run the Global Interactive activity.

#### **Variables**

The task associated to a Global Interactive activity can access the following types of variables:

- predefined variables
- arguments
- local variables

#### **Transitions**

There are no transitions to or from a Global Interactive activity.

### Defining the Task of a Global Interactive Activity

You must define the logic of a Global Interactive activity in the main task.

There are different options to implement the tasks of a Global Interactive activity.

The task of a Global Interactive activity can have the following implementation types:

- Method
- Screenflow
- Show Workload
- Show Dashboard
- Edit Business Rules

The task of a Global Interactive activity with instance access can have the following implementation types:

- Method
- Component
- Screenflow
- Show Process Image
- Display Instance Variables (only if the property Use activity instance for presentation is selected)

The default implementation type is method.

For more information about the different implementation types, see [Flow Object Tasks](#) on page 101.

### Viewing a Global Interactive Activity in Workspace

Describes how Workspace displays Global Interactive activities.

You can view and process Global Interactive activities in WorkSpace if you have the required role. Global Interactive activities appear in the Applications panel. To run a Global Interactive activity click on the link in the Applications panel.

According to the WorkSpace configuration the main task of the Global Interactive activity displays in one of the following ways:

- A dialog
- A maximized dialog
- A pop-up window

### Running a Global Interactive

Describes how the Process Execution Engine processes a Global Interactive activity.

The Process Execution Engine runs the application associated to the Global Interactive activity when the user triggers it.

If the Global Interactive does not have instance access then the running of this activity is completely independent from the process flow.

## Artifacts

This section provides information about the artifacts you can use to design a process.

### Measurement Mark

Measurement marks are checkpoints in the process to measure time or business variables.

Measurement Marks allow you to measure performance metrics, workload metrics and business variables on an event driven basis.

The Measurement Mark gathers these metrics when the instance flows through the transition and immediately after it stores this information to the Engine database. Then the BAM Updater processes this information and adds it to the BAM and Data Mart databases.

You can only associate business variables of type measurement to a Measurement Mark.

Use a Snapshot Start and Stop Measurement Mark to measure:

- the value of a business variable when it leaves an activity.
- the instance flow between flow objects.
- the performance in the previous flow object.

Use a combination of Snapshot Start Measurement Mark and Snapshot Stop Measurement Mark to measure:

- the workload of a set of flow objects.
- the performance of a set of flow objects.
- the value of a business variable for a set of flow objects.
- the elapsed time for a certain part of the process.

The following table describes the available types of Measurement Marks:

Measurement Mark Type	Description
Snapshot Start	Snapshot Start Measurement Marks indicate the start of a measurement. They record the time and the initial value of the associated Business Variables.
Snapshot Stop	Snapshot Stop Measurement Marks are always associated to a Snapshot Start Measurement Mark. They indicate the end of the measurement.
Snapshot Start and Stop	Measure predefined an associated business variables when the instance flows through the transitions.

### Adding a Measurement Mark

When you add a Measurement Mark, you should take into account the following considerations.

You can add a Measurement Mark to a transition to measure predefined variables and business variables when the instance flows through that transition.

### Roles

Measurement activities can appear either in automatic roles or in the user defined role types. However, if the Measurement activity is in a user-defined role, it will not appear in WorkSpace.

### Transitions

Measurement marks are associated to one transition.

When the Engine routes the instance through that transition it performs all the checkpoints associated with the transition.

### Viewing a Measurement Mark in WorkSpace

Describes how you can view a Measurement Mark in WorkSpace.

Measurement Marks do not have a user interface because the engine runs them without the involvement of a user. You cannot process Measurement Marks activities.

You can view information about the processing of a Measurement Marks activity in the audit trail.

You can view the Measurement Mark gathered using BAM Dashboards.

### Running a Measurement Mark

Describes how the Process Execution Engine processes a Measurement Mark.

When an instance flows through a transition with a Measurement Mark, the Process Execution Engine performs all the checkpoints that correspond to the performance variables and business variables associated to the Measurement Mark.

## Adding a Flow Object

The following procedure shows you how to add a flow object to your process.

To add a flow object:

1. Select a flow object from the modeling element palette, located to the right of the process panel.
2. Move the flow object to the place where you want to add it in the process, and click to drop it there.  
If the selected flow object requires to be included in the process flow, then when you approximate it to a transition Studio highlights the transition. If you click while the transition is highlighted Studio automatically adds the required transitions to connect the flow object to the process flow.  
A window to configure the flow object properties appears.
3. Enter a name to identify the flow object in the **Name** field.
4. Click **OK**.  
The selected flow object appears in the process.

### Flow Object Naming Conventions

Providing descriptive names for your flow object allows your process to be self-documented.

Oracle recommends that you name your activity descriptively with a verb followed by a noun specifying the function of the activity within the process. Some examples of useful activity names are: *Create Order*, *Ship Product*, and *Check Credit*.



**Note:** After you define an activity name, you cannot change it. However, you can change the activity label displayed to end users.



## Configuring a Flow Object Properties

The following procedure shows you how to configure the properties of a flow object.

To configure the properties of a flow object:

1. Right-click on the flow object.
2. Select **Properties**.  
A window displaying the properties for the selected flow object appears.
3. Edit the value of the flow object properties.  
For a complete description of these properties, see [Flow Objects Property Reference](#) on page 81.
4. Click **OK**.

## Flow Objects Property Reference

The following sections describe the properties you can specify for the different types of flow objects.

### General Flow Object Property Reference

This reference provides detailed information on properties shared by all flow objects.

#### Activity ID

In the **Activity ID** section, you can configure the following properties for a flow object.

Property	Description
Activity ID	Specifies the ID Oracle BPM Studio and the Process Execution Engine use internally to identify the activity.
Name	Defines a label that describes the name of the activity. This label is used within process diagrams and WorkSpace.
Description	Provides a general description of the Activity.

#### Image

In the **Activity ID** section, you can configure the following properties for a flow object.

Property	Description
Invert Color Palette When Image is Selected	Specifies if the process designer highlights this image when you select it. To highlight the image the process designer inverts its color palette. This property is available only for custom images.
Custom Image	Allows you to specify a custom image for this activity.
Reset Image	Resets the image to the default icon based on your current theme.

### Interactive Activity Property Reference

This reference provides information on how to configure an Interactive activity.

#### Runtime

In the **Runtime** section, you can configure the following properties for an Interactive activity.

Property	Description
Suspendable	Specifies if you can suspend a process instance sitting in this activity. When you suspend a process instance, the process execution engine

Property	Description
	ignores its due transitions and the process deadline. When you resume the process instance, the process execution engine recalculates due transitions and process deadlines, adding the time it spent suspended.
User Selects Transition	<p>Specifies if the user can decide which transition to follow. The user can choose from a list of all the transitions that apply to the current process instance.</p> <p>If you enable this property, the process designer allows you to add multiple unconditional transitions to an Interactive activity.</p>
Auto Complete	Specifies if the instance flows to the next activity immediately after the user executes the mandatory main task or non read-only task.
Abortable	Specifies if the user can eliminate instances from the process. When you eliminate a process instance it flows directly to the end of the process.
Assignable	<p>Specifies if the user can assign the instance in this activity to another user. According to the permissions and categories granted to the user, the following assignment operations are available:</p> <ul style="list-style-type: none"> <li>• Reassign</li> <li>• Peer assign</li> <li>• Escalate</li> <li>• Delegate</li> </ul>

### Advanced

In the **Advanced** section, you can configure the following properties for an Interactive activity.

Property	Description
Unlimited Concurrent Executions	Specifies if the amount of instances that can sit in an Interactive activity at a given time is unlimited.
Number of Concurrent Executions	Specifies the number of instances that can sit at the Interactive activity at a given time. If the number of instances exceeds this limit, the remaining instances are queued.
Generate Events	<p>Specifies if running this activity generates events. The information in the audit trail and BAM tables is based on these events. Possible values are:</p> <ul style="list-style-type: none"> <li>• Default</li> <li>• Generate events</li> <li>• Do not generate events</li> </ul>

### Decision Activity Property Reference

This reference provides information on how to configure a Decision activity.

### Runtime

In the **Runtime** section, you can configure the following properties for a Decision activity.

Property	Description
Suspendable	Specifies if you can suspend a process instance sitting in this activity. When you suspend a process instance, the process execution engine ignores its due transitions and the process deadline. When you resume the process instance, the process execution engine recalculates due transitions and process deadlines, adding the time it spent suspended.
User Selects Transition	<p>Specifies if the user can decide which transition to follow. The user can choose from a list of all the transitions that apply to the current process instance.</p> <p>If you enable this property, the process designer allows you to add multiple unconditional transitions to a Decision activity.</p>
Auto Complete	This property is not available for Decision activities.
Abortable	Specifies if the user can eliminate instances from the process. When you eliminate a process instance it flows directly to the end of the process.
Assignable	<p>Specifies if the user can assign the instance in this activity to another user. According to the permissions and categories granted to the user, the following assignment operations are available:</p> <ul style="list-style-type: none"> <li>• Reassign</li> <li>• Peer assign</li> <li>• Escalate</li> <li>• Delegate</li> </ul>

### Advanced

In the **Advanced** section, you can configure the following properties for a Decision activity.

Property	Description
Unlimited Concurrent Executions	Specifies if the amount of instances that can sit in a Decision activity at a given time is unlimited.
Number of Concurrent Executions	Specifies the number of instances that can sit at the Decision activity at a given time. If the number of instances exceeds this limit, the remaining instances are queued.
Generate Events	<p>Specifies if running this activity generates events. The information in the audit trail and BAM tables is based on these events. Possible values are:</p> <ul style="list-style-type: none"> <li>• Default</li> <li>• Generate events</li> <li>• Do not generate events</li> </ul>

### Automatic Activity Property Reference

This reference provides information on how to configure an Automatic activity.

### Runtime

In the **Runtime** section, you can configure the following properties for an Automatic activity.

Property	Description
Enable Loop Condition	Specifies if the process execution engine should run this activity successively until the defined condition is met.
Evaluation Order	Specifies if the process execution engine evaluates the condition before or after running the Automatic activity. Possible values are: <ul style="list-style-type: none"> <li>• Before</li> <li>• After</li> </ul>
Condition	Specifies the condition that determines if the process execution engine keeps running the loop. You must specify the condition using PBL.

### Advanced

In the **Advanced** section, you can configure the following properties for an Automatic activity.

Property	Description
Unlimited Concurrent Executions	Specifies if the amount of instances that can sit in an Automatic activity at a given time is unlimited.
Maximum Number of Instances	Specifies the number of instances that can sit at the Automatic activity at a given time. If the number of instances exceeds this limit, the remaining instances are queued.
Generate Events	Specifies if running this activity generates events. The information in the audit trail and BAM tables is based on these events. Possible values are: <ul style="list-style-type: none"> <li>• Default</li> <li>• Generate events</li> <li>• Do not generate events</li> </ul>

### Subflow Activity Property Reference

This reference provides information on how to configure a Subflow activity.

### Related Process

In the **Related Process** section, you can configure the following properties for a Subflow activity.

Property	Description
Dynamic Process Invocation	Specifies if the related process is a process interface.
Related Process	Specifies the process that is used as a subprocess. If you select Dynamic Process Invocations the related process must be a process interface.
Argument Set Name	Specifies the argument set the Subflow activity uses to invoke the subprocess.

### Advanced

In the **General** section, you can configure the following properties for a Subflow activity.

Property	Description
Attachments can be Visible to Related Process	Specifies if the files attached to the process instance are visible from the subprocess.

Property	Description
Process Notification Immediately	Specifies if the Engine processes the notification to the Subflow activity and the argument mapping, in the same transaction.
Generate Events	<p>Specifies if running this activity generates events. The information in the audit trail and BAM tables is based on these events. Possible values are:</p> <ul style="list-style-type: none"> <li>• Default</li> <li>• Generate events</li> <li>• Do not generate events</li> </ul>

### Process Creation Property Reference

This reference provides information on how to configure a Process Creation activity.

#### Related Process

In the **Related Process** section, you can configure the following properties for a Process Creation activity.

Property	Description
Dynamic Process Invocation	Specifies if the related process is a process interface.
Related Process Name	Specifies the process that is used as a subprocess. If you select Dynamic Process Invocations the related process must be a process interface.
Argument Set Name	Specifies the argument set the Subflow activity uses to invoke the subprocess.

#### Advanced

In the **Advanced** section, you can configure the following properties for a Process Creation activity.

Property	Description
Attachments can be Visible to Related Process	Specifies if the files attached to the process instance are visible from the subprocess.
Keep Relation With Child Process	Specifies if the Process Creation activity keeps track of the instances it creates in the subprocess.
Generate Events	<p>Specifies if running this activity generates events. The information in the audit trail and BAM tables is based on these events. Possible values are:</p> <ul style="list-style-type: none"> <li>• Default</li> <li>• Generate events</li> <li>• Do not generate events</li> </ul>

### Termination Wait Activity Property Reference

This reference provides information on how to configure a Termination Wait activity.

#### Runtime

In the **Runtime** section, you can configure the following properties for a Termination Wait activity.

Property	Description
Process Creation Activity	Specifies the Process Creation activity associated to this Termination Wait Activity.

### Advanced

In the **Advanced** section, you can configure the following properties for a Termination Wait activity.

Property	Description
Generate Events	Specifies if running this activity generates events. The information in the audit trail and BAM tables is based on these events. Possible values are: <ul style="list-style-type: none"> <li>• Default</li> <li>• Generate events</li> <li>• Do not generate events</li> </ul>
Process Notification Immediately	Specifies if the Engine processes the notification to the Process Creation activity and the argument mapping, in the same transaction.

### Grab Activity Property Reference

This reference provides information on how to configure a Grab activity.

### Runtime

In the **Runtime** section, you can configure the following properties for a Grab activity.

Property	Description
Grab Type	Specifies the type of Grab Possible values are: <ul style="list-style-type: none"> <li>• Defined</li> <li>• From all</li> <li>• From all/To all/</li> </ul> For more information, see <a href="#">Grab Types</a> on page 87.
Suspendable	Specifies if you can suspend a process instance sitting in this activity. When you suspend a process instance, the process execution engine ignores its due transitions and the process deadline. When you resume the process instance, the process execution engine recalculates due transitions and process deadlines, adding the time it spent suspended.
User Selects Transitions	Specifies if the user can decide which transition to follow. The user can choose from a list of all the transitions that apply to the current process instance. <p>If you enable this property, the process designer allows you to add multiple unconditional transitions to a Grab activity.</p>
Abortable	Specifies if the user can eliminate instances from the process. When you eliminate a process instance it flows directly to the end of the process.

## Grab Types

Type	Description
Defined	You must define an inbound transition to specify from which flow objects you can grab an instance, and an outbound transition to specify to which flow objects you can redirect the grabbed instance.
From All	The Grab activity has an implicit inbound transition from all the flow objects in the process flow. You must define an outbound transition to specify to which flow objects you can redirect the grabbed instance.
From All/To All	The Grab activity has implicit inbound and outbound transitions to all the flow objects in the process flow. You can grab an instance from any flow object in the process flow and you can redirect the instance to any flow object in the process flow.

## Advanced

In the **Advanced** section, you can configure the following properties for a Grab activity.

Property	Description
Generate Events	Specifies if running this activity generates events. The information in the audit trail and BAM tables is based on these events. Possible values are: <ul style="list-style-type: none"> <li>• Default</li> <li>• Generate events</li> <li>• Do not generate events</li> </ul>

## Conditional Gateway Property Reference

You can only configure general properties for a Conditional gateway.

You can configure the properties under **Activity Id** and **Image** sections for a Conditional gateway.

## Split Gateway Property Reference

This reference provides information on how to configure a Split gateway.

## Advanced

In the **Advanced** section, you can configure the following properties for a Split gateway.

Property	Description
Generate Independent Copies	Specifies if the instance copies the split gateway generates are independent. Independent instance copies do not share the value of process variables during the split circuit. If you select this option you need to merge the data of the different instance copies in the join gateway.
Generate Events	Specifies if running this activity generates events. The information in the audit trail and BAM tables is based on these events. Possible values are: <ul style="list-style-type: none"> <li>• Default</li> <li>• Generate events</li> <li>• Do not generate events</li> </ul>

### Or-Split Gateway Property Reference

This reference provides information on how to configure an Or-Split gateway.

#### Advanced

In the **Advanced** section, you can configure the following properties for an Or-Split gateway.

Property	Description
Generate Independent Copies	Specifies if the instance copies the split gateway generates are independent. Independent instance copies do not share the value of process variables during the split circuit. If you select this option you need to merge the data of the different instance copies in the join gateway.
Generate Events	Specifies if running this activity generates events. The information in the audit trail and BAM tables is based on these events. Possible values are: <ul style="list-style-type: none"> <li>• Default</li> <li>• Generate events</li> <li>• Do not generate events</li> </ul>

### Multiple Gateway Property Reference

This reference provides information on how to configure a Multiple Gateway.

#### Advanced

In the **Advanced** section, you can configure the following properties for a Multiple gateway.

Property	Description
Maximum Number of Simultaneous Copies	Specifies the number of copies that can flow simultaneously within the circuit. If the amount of copies exceeds this number, the remaining copies are queued until one of the copies in the circuit reaches the Join event or is aborted.
Generate Events	Specifies if running this activity generates events. The information in the audit trail and BAM tables is based on these events. Possible values are: <ul style="list-style-type: none"> <li>• Default</li> <li>• Generate events</li> <li>• Do not generate events</li> </ul>


### Join Property Reference

This reference provides information on how to configure a Join gateway.

#### Advanced

In the **Advanced** section, you can configure the following properties for a Join gateway.



Property	Description
Amount of copies to wait to release	<p>Specifies the number of copies to wait for before the instance resumes the process flow. When this number of copies reaches the join, the engine terminates the remaining copies that still in the circuit.</p> <p> <b>Note:</b> If you set this property value to zero, the instance waits for all the copies in the circuit to reach the join before resuming the process flow.</p>
Generate events	<p>Specifies if running this activity generates events. The information in the audit trail and BAM tables is based on these events. Possible values are:</p> <ul style="list-style-type: none"> <li>• Default</li> <li>• Generate events</li> <li>• Do not generate events</li> </ul>

### Message Wait Property Reference

This reference provides information on how to configure a Message Wait event.

#### Advanced

In the **Advanced** section, you can configure the following properties for a Message Wait event.

Property	Description
Process Creation Activity	Specifies the Process Creation activity associated to this Message Wait event. This property is valid only when the notifies comes from a child process.
Wait for (event type)	<p>Specifies the type of event that triggers the Message Wait event. Possible values are:</p> <ul style="list-style-type: none"> <li>• Parent Process</li> <li>• Child Process</li> <li>• External Event</li> </ul> <p>For more information, see <a href="#">Message Wait Event Types</a> on page 70.</p>
Allows Interruptions	Specifies if the Message Wait event behaves like a listener that waits for an event to occur to pull out the affected instances from the main process flow and process them using the defined Message Wait flow. If you select this property, the icon of the Message Wait event changes.
Process notification Immediately	Specifies if the Engine processes the notification to the Message Wait event and the argument mapping, in the same transaction.

#### Advanced

In the **Advanced** section, you can configure the following properties for a Message Wait event.

Property	Description
Generate Events	<p>Specifies if running this activity generates events. The information in the audit trail and BAM tables is based on these events. Possible values are:</p> <ul style="list-style-type: none"> <li>• Default</li> </ul>

Property	Description
	<ul style="list-style-type: none"> <li>• Generate events</li> <li>• Do not generate events</li> </ul>

### Send Message Event Property Reference

This reference provides information on how to configure a Send Message event.

#### Notification

In the **Notification** section, you can configure the following properties for a Send Message event.

Property	Description
Related Process Name	Specifies the process to notify.
Notifies	Displays the type of the Message Wait event.
Notification Target	Specifies the Message Wait event that receives the notification in the related process.
Argument Set Name	Specifies the Argument Set of the Message Wait event, that this Send Message uses to notify the Message wait event.
Process Creation Activity	This property is available if the related process is a subprocess of the current process. It specifies the Process Creation that created the instance in the subprocess.

#### Advanced

In the **Advanced** section, you can configure the following properties for a Message Wait event.

Property	Description
Generate Events	<p>Specifies if running this activity generates events. The information in the audit trail and BAM tables is based on these events. Possible values are:</p> <ul style="list-style-type: none"> <li>• Default</li> <li>• Generate events</li> <li>• Do not generate events</li> </ul>

### Begin Event Property Reference

This reference provides information on how to configure a Begin event.

#### General

In the **General** section, you can configure the following properties for an End event.

Property	Description
Unlimited Concurrent Process Instances	Specifies if the amount of instances that can sit in a Begin event at a given time is unlimited.
Action Performed When Limit is Reached	Specifies how to handle concurrent executions when the concurrent process instances exceed the defined limit.

## Advanced

In the **Advanced** section, you can configure the following properties for an End event.

Property	Description
Generate Events	Specifies if running this activity generates events. The information in the audit trail and BAM tables is based on these events. Possible values are: <ul style="list-style-type: none"> <li>• Default</li> <li>• Generate events</li> <li>• Do not generate events</li> </ul>

## End Event Property Reference

This reference provides information on how to configure an End event.

## Advanced

In the **Advanced** section, you can configure the following properties for an End event.

Property	Description
Generate Events	Specifies if running this activity generates events. The information in the audit trail and BAM tables is based on these events. Possible values are: <ul style="list-style-type: none"> <li>• Default</li> <li>• Generate events</li> <li>• Do not generate events</li> </ul>

## Timer Property Reference

This reference provides information on how to configure a Timer event.

## Runtime

In the **Runtime** section, you can configure the following properties for an End event.

Property	Description
Type	Possible values are: <ul style="list-style-type: none"> <li>• Schedule based</li> <li>• Interval expression</li> <li>• Interval constant</li> </ul>
Use calendar rules	Specifies if calendar rules affect the carrying out of this activity.
On holidays run on	Specifies the behaviour when the scheduled time is a holiday. This property is available only when you select the property <b>Use calendar rules</b> . Possible values are: <ul style="list-style-type: none"> <li>• Next working day and same time</li> <li>• Next scheduled based time</li> </ul>

**Schedule Based**

Property	Description
Frequency	Specifies when to run this activity. Possible values are: <ul style="list-style-type: none"> <li>• Daily</li> <li>• Weekly</li> <li>• Monthly</li> </ul>
When	Specifies the exact time to run this activity. The fields to define the time vary according to the frequency you select. You can add multiple times.

**Interval Expression**

You must define an expression in PBL that determines when to run this activity.

**Interval Constant**

You must specify an interval to determines the rate at which to run this activity

**Compensate Event Property Reference**

This reference provides information on how to configure a Compensate activity.

**Compensate**

Property	Description
Activity to compensate	Specifies an activity to revert when the Compensate event is associated to a group. If you do not select an activity the Compensate event reverts all the activities included in the group.
Generate Events	Specifies if running this activity generates events. The information in the audit trail and BAM tables is based on these events. Possible values are: <ul style="list-style-type: none"> <li>• Default</li> <li>• Generate events</li> <li>• Do not generate events</li> </ul>

**Global Creation Property Reference**

This reference provides information on how to configure a Global Creation activity.

**Runtime**

In the **Runtime** section, you can configure the following properties for a Global Creation activity.

Property	Description
Auto Complete	Specifies if the instance flows to the next activity immediately after the user executes the mandatory main task or non read-only task.
Argument Set Name	Specifies the argument set to use when invoking the Begin event.

**Global Automatic Property Reference**

This reference provides information on how to configure a Global Automatic activity.

## Runtime

In the **Runtime** section, you can configure the following properties for an Interactive activity.

Property	Description
Global Automatic Type	Specifies the type of this Global Automatic activity. Possible values are: <ul style="list-style-type: none"> <li>• Polling by interval</li> <li>• Execute when an event occurs</li> <li>• Automatic schedule</li> <li>• Automatic JMS listener</li> <li>• Automatic JMX listener</li> </ul>
Stop Running When Process is Deprecated	Specifies if this activity stops running when you deploy a newer version of the process.

## Global Automatic Types

The following table describes the different types you can select for a Global Automatic.

Property	Description
Polling by interval	The Engine waits for a time interval to run the Global Automatic activity.
Executes when an event occurs	The Engine waits for an event to occur to run the Global Automatic activity.
Automatic schedule	The Engine runs the Global Automatic activity at the times you define in a schedule.
Automatic JMS listener	The Engine runs the Global Automatic activity when a JMS message arrives to the specified queue or topic.
Automatic JMX listener	The Engine runs the Global Automatic activity when it receives a JMX message.

## Polling by Interval

Property	Description
Use Calendar Rules	Specifies if the Engine uses calendar rules to calculate the interval. If selected the interval only includes working days.
Wait Interval	Specifies the time to wait between successive executions of this activity.

## Executes When an Event Occurs

Property	Description
Listener Class	A Component that implements the interface <code>Fuego.Lib.ServerEventSource</code> .
Component Event Type	Specifies the type of event to listen to. The Component Event Types are defined in the Listener Class.

**Automatic Schedule**

Property	Description
Run on Holidays	Specifies if the engine runs this activity when the scheduled time corresponds to a holiday.
Frequency	Specifies when to run this activity. Possible values are: <ul style="list-style-type: none"> <li>• Daily</li> <li>• Weekly</li> <li>• Monthly</li> </ul>
When	Specifies the exact time to run this activity. The fields to define the time vary according to the frequency you select.

**Automatic JMS Listener**

Property	Description
JMS Configuration	An external resource of type JMS.
Message Selector	A message selector allows a JMS consumer select some of the messages it receives from a particular topic or queue using a defined criteria. A message selector uses message properties and headers to define conditional expressions. These expressions use Boolean logic to define which messages should be delivered to the JMS client .

**Automatic JMX Listener**

Property	Description
JMX Configuration	An external resource of type JMX. The Global Automatic activity uses this external resource to connect to the MBean Server.
JMX Component	The introspected JMX notification the Global Automatic activity listens to.
Object Name	The ObjectName used to locate the MBean that emits the notification.

**Global Interactive Activity Property Reference**

This reference provides information on how to configure a Global Interactive activity.

**Runtime**

In the **Runtime** section, you can configure the following properties for an Interactive activity.

Property	Description
Has Instance Access	Specifies if the Global Interactive activity can access the instance information.
Use Activity for Instance Presentation	Specifies if the Global Interactive activity is used to display the instance information in the Instance Detail panel.
Generate Events	Specifies if running this activity generates events. The information in the audit trail and BAM tables is based on these events. Possible values are: <ul style="list-style-type: none"> <li>• Default</li> </ul>

Property	Description
	<ul style="list-style-type: none"> <li>• Generate events</li> <li>• Do not generate events</li> </ul>

### Measurement Mark Property Reference

This reference provides information on how to configure a Measurement Mark.














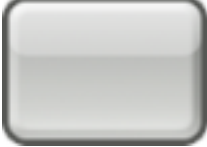


You can configure the following properties for a Measurement Mark.










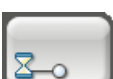






Property	Description
Measurement mark type	Possible values are: <ul style="list-style-type: none"> <li>• Snapshot start</li> <li>• Snapshot stop</li> <li>• Snapshot start-stop</li> </ul>
ID	Specifies the ID that identifies this Measurement Mark.
Name	Specifies the name that identifies Measurement Marks of type start and start-stop. You can localize this name.
Start Measurement	Specifies the Measurement Mark of type start associated to a Measurement Mark of type stop.
Description	Provides a general description of the Measurement Mark. You can localize this description.
Project variables	Specifies the project variables the Measurement Mark monitors. You can only monitor business variables of type dimension.

### Flow Object Icon Reference





















The following reference table shows the flow object icons for each of the process themes supported in Oracle BPM Studio.

#### Activities





















Activity Name	Classic Icon	BPMN Icon	BPMN Color Icon	UML Icon
Interactive				
Decision				
Automatic				
Group				

Activity Name	Classic Icon	BPMN Icon	BPMN Color Icon	UML Icon
Subflow				
Process Creation				
Termination Wait				
Grab				

### Gateways

Activity Name	Classic Icon	BPMN Icon	BPMN Color Icon	UML Icon
Conditional				
Split				
Or-Split				
Multiple				
Join				










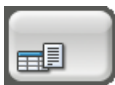


### Events

Activity Name	Classic Icon	BPMN Icon	BPMN Color Icon	UML Icon
Message Wait				
Send Message				
Timer				
Compensate				
Begin				







Activity Name	Classic Icon	BPMN Icon	BPMN Color Icon	UML Icon
End				

### Global Activities

Activity Name	Classic Icon	BPMN Icon	BPMN Color Icon	UML Icon
Global Creation				
Global Automatic				
Global Interactive				

### Artifacts

Activity Name	Classic Icon	BPMN Icon	BPMN Color Icon	UML Icon
Measurement Mark				

## Groups

A Group is a compound activity. It is composed of a set of flow elements that may include other Groups.

Groups are primarily used to provide exception and compensation handlers to a group of activities. They are also commonly used to handle timeouts for a group of activities.

A Group can be formed by any flow element (also called leaf activity) with the exception of Global Interactive, Global Creation, Global Automatic, Begin and End activities. Also keep in mind that a group cannot contain only a Split or a Join gateway.

### Using Groups

Groups are useful in the following scenarios:

- You need to define a due interval within which a group of activities must be completed.
- You need to manage a specific exception that can occur in any activity within a group. Instead of handling the exception in each activity, you can define the group and handle the exception from within it.
- You need to reverse (compensate) a certain situation that involves more than one activity to be rolled back.
- You need to manage a group of activities as a unique transaction (atomic group)
- You need to simplify the design. Groups help you better visualize the process design since you can collapse a set of activities into one group element.

### Compensating Work in a Group

If you want to undo the actions in a group, you should raise an exception and trigger a Compensate event.

## Creating a Group

The following procedures outline the process for creating a Group.

1. Determine which flow elements you want to include in the Group.
2. Select the Group icon in the toolbar.
3. Select all the flow elements within the design that should be moved to the group.
4. Right-click and select Create group with selection.

Transitions between the elements will remain and two new ones will be generated: a unique incoming transition to the group and a unique outgoing transition from the group.

The **Group** window appears.

5. Provide a name and optional description for the Group.
6. Edit the Group properties as necessary.

See [Group Properties](#) on page 100 for more information.

7. Click **OK**.

The Group is created and appears as dotted lines around the selected activities.

## Groups and Transitions

Transitions to and from Groups behave differently than Transitions to and from individual activities.

### Due Transitions

A due transition can be specified for 'leaf' activities as well as for 'group' activities.

A group can have a due transition that applies to the instance in any activity within the group. In such case, due time will begin running as soon as the instance arrives at the first activity within the group (entry point).

To define which calendar rule to apply, the engine looks for the organizational unit where the process of the instance being executed has been deployed. If no calendar rule is defined for the organizational unit, the engine keeps looking in the upper levels of the organizational hierarchy until it finds a calendar rule to apply. If no calendar rule is found, it is assumed that all days are working days. The Engine will also take into account the calendar rule set for the organizational unit where the process is deployed and the activity role where the instance is running. The calendar rule set at role level is first evaluated by the Engine and overrides rules defined for the organizational unit, if any are defined. Refer to Calendar Rules for further information.

Group due transitions override any single activity's (within the group) due transition. If a group has a due transition and it exists inside another group that has a due transition, the instance is ruled by the outermost group's due transition if both transitions expire at the same time. For example, between an activity due transition and the group that contains the activity, due transition, the instance will be routed according to the group due transition

When an activity task is being executed over an instance, the task timeout is set to the shortest due interval of:

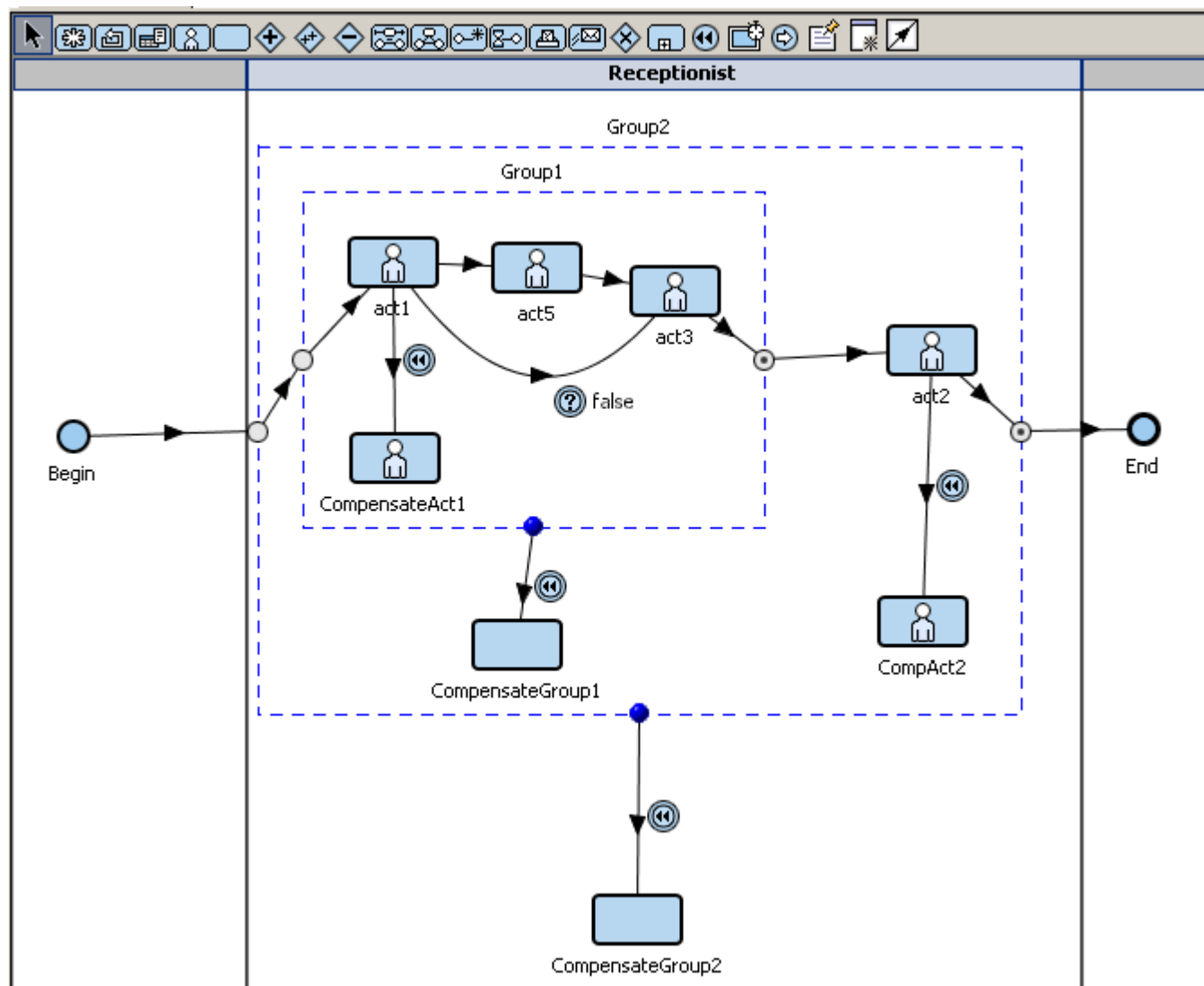
- The task timeout interval (defined as an engine property),
- PBL-Method timeout interval that is currently being executed (i.e., the PBL-Method that is being executed within the list of PBL-Methods that might be connected with relay-to),
- The activity due transition interval,
- Each nesting group due transition (if there are multiple nested groups, all due transition are evaluated),
- The process deadline.

## Compensate Transitions

Each leaf activity can have its own compensate flow. In addition, the group can also have its own compensate flow. The group's compensate transition can send the instance to a leaf activity or to a group activity that will perform any necessary action to compensate the situation. This group represents the compensate flow.

Within the group's compensate flow, you might not reference a specific inner compensation (compensation defined in some inner activity). Therefore, the engine will call all the compensations defined in the group in the inverted order in which they are executed. If it has reference to an inner compensation, the engine will call only this compensation.

In the following figure, Group 1 is an inner group of Group2. Moreover, Group1 has activities act1, act3 and act5 within it.



- If act1 needs to be compensated, the CompensateAct1 activity will be executed.
- If act3 needs to be compensated, as there is no compensate transition going out of this activity, no compensation is performed.
- If no compensate transition is defined for a Group, then all inner activities compensations are performed. They are performed in reverse order of the process flow. In the example, if there were no compensate transition for Group2, inner compensations are executed: CompAct2 and then CompensateGroup1 in this order.

A group can also be defined by grouping activities inside the compensate flow or exception flow. In this case, no exit point for the group will exist. See [Compensate Transition](#) on page 119.

### Exception Transitions

Groups can call and manage their own exception flow. See [Exception Transition](#) on page 118 for more information.

## Groups and Grab Activities

An instance can be grabbed from any activity, regardless of whether the activity is grouped or not.

While grabbed, the instance will remain as grabbed in the grab activity, as in the following scenarios:

- A Grab activity's task is executed over the instance, or
- The instance is ungrabbed, or
- The instance is sent to an activity different from the source activity, the source group's property will rule the operation (i.e., due transition expiration, exception handling, compensation handling).

The source activity's due time is removed when the instance is grabbed. On the other hand, the process instance deadline is kept active. Moreover, the due time for groups nesting the source activity is also kept active (in the event that the activity is within a group that is actually within another group).

When a grabbed instance is ungrabbed, from that moment on, the instance can be processed as usual in the source activity (activity from which it was grabbed).



**Note:** Once an instance has been grabbed, it can be sent to an activity different from the source activity but only within the same Group.

## Group Properties

### General Properties


The following table lists the general properties.

Property	Description
Is Atomic	<p>A Group can be flagged as Atomic in order to be executed as a single transaction. An atomic group is defined as a group of automatic activities all executed in one transaction. For a group to become atomic, all its Sub-Groups must be flagged as atomic. An atomic group cannot include Interactive activities or external notifications.</p> <p>Activities inside an atomic group can belong to different roles.</p> <p>An atomic group can contain ONLY:</p> <ul style="list-style-type: none"> <li>• Automatic activities.</li> <li>• Split/Join gateways.</li> <li>• Multiple/Join gateways.</li> <li>• Send Message/Message Wait events (only synchronization between copies.)</li> </ul> <p>If any other activity (such as Interactive) is dropped into a group, you will be warned about it and the action will be automatically undone.</p>

Property	Description
	<p>An atomic group can contain other atomic groups. However, it cannot contain non-atomic groups.</p> <p>If a non-atomic group is dropped into an atomic group, you will be warned about it and the action will be undone.</p> <p>Atomic groups cannot handle exceptions or compensations within the group. Exceptions that occur inside the atomic group have to be handled by the group exception flow or any outer group.</p>

### Advanced Properties

The following table lists the general properties.

Property	Description
Generate Events	<p>Defines how Auditing Events are generated for the Group. See <a href="#">Auditing</a> on page 221.</p> <p> <b>Note:</b> If the group does not generate events, it will not be listed in the Audit Trail, but the activities within it, if they do generate events, will appear</p>

## Flow Object Tasks

The following topics provide information about using Tasks within Oracle BPM Studio.

### What is a Task?

A task consists of one or more actions that need to be executed in order to achieve a flow object's goal. Each type of flow object can contain only one task, multiple tasks or no tasks at all.

There is a Main task and, additionally, some Optional tasks can be added. Flow objects that require human intervention can have optional tasks, such as the Interactive activity and the Grab activity. The Main task is automatically generated and the optional tasks have to be added and defined by the developer. The associated PBL-Methods will also be generated. The first task in the list is the Main task, and the order of all Optional tasks is the order in which they appear on the list within the dialog box where they are defined.

Some tasks are mandatory, which means that they should be processed before the instance can be sent to the next flow object in the business service. Additionally, a task can be defined as repeatable, which means that this task can be performed as many times as necessary until the instance moves to the next flow object.

Tasks appear in a list in WorkSpace. A WorkSpace user selects the task to run in the order that he or she chooses.

### Implementation Type

Each Task contains an Implementation Type which defines how the Task is implemented and how it behaves within a business process. You can define the implementation type within a Task's properties. Not all Implementation Types are available within each task.

## Using Tasks Within Flow Objects

Flow Object	Number of Tasks	Automatically generated	Description
Begin	None	None	.
End	None	None	.
Global Creation	1	Yes	.
Global Automatic	1	Yes	.
Global	1	If it is a Method	This can be a <b>Method</b> and therefore it is created with the same name as the flow object.
Interactive	Multiple: Main and optional tasks.	No	The developer should define as many tasks as required in the flow object.
Automatic	Main one	No	The developer should define as many tasks as required in the flow object. No repeatable or mandatory properties will appear for this type of flow object.
Split	1	Yes	.
Split N	1	Yes	.
Join	1	Yes	.
Grab	Multiple: Main and optional tasks.	No	The developer should define as many tasks as required in the flow object.
Subflow	None	None	.
Process Creation	None	None	.
Termination Wait	None	None	.
Message Wait	None	None	.
Process Notification	None	None	.

## Tasks Types

The topics in the section describe the different Implementation Types within an Oracle BPM Task. Different flow objects can contain different Implementation Types.

### Method Tasks

The Method Task allows you to define a PBL Method to perform the Task for a flow object.

A Method is a high-level scripting language used to define the business rules and the logic of flow object types and certain transitions within a process. Each flow object type initiates a very specific action and the PBL-Method provides the script for this action. PBL-Method includes statements that integrate variables, expressions, operators and components.

## Using the Method Editor

The Method Editor is a free-form text editor that allows you to copy, cut and paste partial or entire PBL-Methods objects, including statements, expressions, arguments, variables, operators and components

## Method Timeout

Timeouts apply to PBL-Method. Timeouts have a default value (5 minutes) that can be redefined using the predefined variable timeout. As well, no matter the defined timeout for each of them, they are all limited to the maximum specified in the Engine property Maximum PBL-Methods timeout.

See [Method Timeout](#) on page 103 for more information.

## Method Execution Results

When Methods are executed, several results may occur. The following information provides a complete list of possible instance behavior according to the result of the PBL-Method. The predefined variable action is used to indicate the Method result.

The Engine acts according to this variable and saves or undoes (commits or rollbacks) the changes.

The action variable is an enumeration that accepts the following valid values:

## Calling Components from Methods

A number of standard components (Library Components) are available at the time of installation for you to use in your process design. You may also add your own Java, SQL, EJB, JNDI, Web Service and BPM Objects.

The syntax for calling a component is the same, irrespectively of the type of component you are using. In BPM skin, it will be as follows:

Syntax: [method name] [component name]

Example:

For example, a method name is calculateTotal and the component name is Pricing .

calculateTotal Pricing

## Method Timeout

A timeout defines the amount of time a Task will wait to complete an action before processing continues.

## Default Value

By default the PBL-Method has the timeout set to 5 minutes. You can change this value using the timeout Predefined Variable.

## Extending the Timeout Value

This method is not recommended for all situations. Instead, it should be used only occasionally. During the PBL-Method execution, several Engine resources are locked, therefore if you extend the timeout for each PBL-Method, the risk of having all the resources locked increases and it may produce a bottleneck.

The syntax is as follows:

```
// timeout is an interval
// Increasing it to 20
minutes timeout = '20m'
```



**Note:** If you set the PBL-Method timeout to be greater than the Engine property 'Maximum PBL-Methods timeout', the PBL-Method will fail at runtime and a Engine log message is generated.

If you set the timeout predefined variable to null, then the Engine property (Maximum PBL-Methods timeout) value applies.

## Timeout Limit

The Administrator can limit the timeout to a maximum for all processes deployed in a specific engine using the engine property Maximum PBL-Methods timeout. Maximum PBL-Methods timeout is a tool for the Administrator to ensure that the Engine resources are enough to serve all deployed processes. If a PBL-Method timeout is greater than the 'Maximum PBL-Methods timeout' property, such PBL-Method will fail.

The Maximum PBL-Methods timeout can be set from Studio Engine Properties as well as from Process Administrator Engine Properties.

## Integrated Components Within a PBL-Method

When you use Components within a PBL-Method, you can set a timeout as one of the component attributes. This timeout applies to the component execution time.

If the PBL-Method runs one or more component, although they can have individual timeouts, the PBL-Method timeout rules the full processing.

For example, the maximum PBL-Method timeout is set to 3 minutes. You define a PBL-Method that runs 2 components (Component A and Component B), and both components have the timeout attribute set to 2 minutes.

Component A begins running and finishes in 1 minute, 45 seconds. Although Component B has an individual timeout set to 2 minutes, its execution will not last more than 1 minute, 15 seconds as passed that time the PBL-Method execution is aborted.

## Method Property Reference

The following sections describe the properties defined in the Main Task window using the Method Implementation Type.

### Properties

The following are available in the properties area. These properties are only available in Interactive Activity.

Property	Description
Repeatable	If true, the user is allowed to run this task more than once
Mandatory	If true, the user must run this task to complete the activity
Read-only	If true, this task cannot modify any instance variable

### Method

Property	Description
Method Name	Defines the name of the method used as the Main Task within this Activity.
Roll-back Method	Defines the method used when a transaction fails and work must be rolled-back.

## Component Tasks

The Component Task allows you to specify a component method as the Task for a flow object.

### Component Task Timeout

A timeout defines the amount of time a Task will wait to complete an action before processing continues.



## Default Timeout

The timeout is set in the Activity task execution timeout that works the same way as the PBL-Methods timeout but applicable to the tasks implemented through components (not through a PBL-Method).

By default the Activity task execution timeout property has the timeout set to 5 minutes. You can change this value if required (minutes:seconds).

## Component Property Reference

The following sections describe the properties defined in the Main Task window using the Component Implementation Type.

### Properties

The following are available in the properties area. These properties are only available in Interactive Activity.

Property	Description
Repeatable	If true, the user is allowed to run this task more than once
Mandatory	If true, the user must run this task to complete the flow object.
Read-only	If true, this task cannot modify any instance variable

### Component Method

Property	Description
flow object Task Execution Timeout	Specifies the timeout for the component call execution (minutes:seconds). After this time occurs, the engine cancels the component execution.
Use Instance Variable	Indicates that the invocation of the method will be applied to the selected variable. By selecting a variable, the Component name is implied and taken from the variable type. A typical example of instance variable to use is a BPM Object instance variable
Component Name	Represents the type of component to be invoked. If an instance variable is selected, this value will be filled out from the variable type and cannot be edited from here. Complete the component name or browse to choose the component within the Project Catalog. If the task corresponds to an Interactive activity, you can only select a component defined to run on the client-side
Component Member	<ul style="list-style-type: none"> <li>• Input</li> <li>• Display</li> </ul>
Use Presentation	
Use Instance Variable	
Argument Mapping	If the method requires any in or out arguments, these arguments have to be mapped to instance variables or predefined variables that contain the value/s to be passed. Additionally, if you are using an instance variable (for example a BPM Object instance var) and

Property	Description
	the component updates any of the attributes within it, you need to explicitly map the BPM Object instance variable to an argument variable. For example, map the BPM Object to the currentComponentInstance variable.

### Procedure Tasks

The Procedure Task allows you to specify a Procedure flow as the Task for a flow object.

### Procedure Property Reference

The following section describes the properties defined in the Main Task window using the Procedure Implementation Type.

### Properties

The following are available in the properties area. These properties are only available in Interactive Activity.

Property	Description
Dynamic Process Invocation	If true, the actual procedure to call is defined at runtime.
Related Procedures	Select the procedure to call. If <b>Dynamic Process Invocation</b> is true, you must select a Procedure interface instead of an actual procedure.
Select Procedure Argument Set	Select the argument set you want to use to invoke the procedure

### Screenflow Tasks

The Screenflow Task allows you to specify a Screenflow as the Task for a flow object.

### Screenflow Property Reference

The following section describes the properties defined in the Main Task window of a Screenflow.

### Properties

The following are available in the properties area. These properties are only available in Interactive Activity.

Property	Description
Repeatable	If true, the user is allowed to run this task more than once
Mandatory	If true, the user must run this task to complete the activity
Read-only	If true, this task cannot modify any instance variable

### Related Screenflow

Property	Description
Name	Indicates the screenflow defined within the project to be executed.
Argument Mapping	Select the argument set you want to use to invoke the screenflow

## External Tasks

External tasks are tasks that are implemented outside the BPM system.

The External task implementation allows you to implement the interactive activity (more precisely the GUI presented to the user) outside the BPM system.

It is a framework to allow adding an external UI interface when executing an interactive task without using any of the BPM system's presentation capabilities (BPM Object Presentations, Screenflows, etc). These tasks are accessible from PAPI/PAPIWS by calling the methods. For further information see the PAPI's JavaDoc documentation.

For example, this is useful if you need to integrate the Engine with a .NET client.

```
prepareExternalActivity( ... )
commitExternalActivity( ... )
```

The framework functions as follows:

1. A page displays all instances (you can get this using PAPI or PAPIWS).
2. You select the instance for execution (this instance is waiting in an interactive activity).
3. To start the execution, the prepare method should be executed. This method can retrieve some information from the instance as well as leveraging retrieval of other information using the BPM system's integration capabilities. Upon successful execution of the prepare method, the instance will remain locked for that participant to prevent other participant from starting the execution for that instance.
4. You can render your own UI (ASP.NET, etc).
5. When this is submitted, the commit method should be executed. This method receives information collected from the external UI and most likely synchronize this data with instance variables. You can also perform transactions against your back-end systems using the BPM system's integration capabilities. Upon successful execution of this method, the lock over the instance will be released and the task will be marked as completed.

When you select the External implementation you have to specify two methods:

- The `prepareExternalActivity()` method can be used to get any values needed by the presentation before actually displaying it.

This method extracts and processes information from the instance variables or BPM Objects, and makes it accessible through its output arguments

- The `commitExternalActivity()` method should be invoked once the user finished with the GUI and processing can continue in the BPM system.

This method completes the task (and the instance if necessary), and maps its input arguments to instance variables.

Both methods can declare input/output arguments that will be passed along whenever they are invoked. As well both methods need to be invoked through PAPI or through PAPI-WebServices.

For example:

The instance arrives to the activity and waits there until someone sends the `prepareExternalActivity()` method to it. It executes the method and answers back a set of arguments (valid values lists or predefined/default values to be used by the GUI). After the user finishes with the GUI (which could be implemented anyway you want), the `commitExternalActivity()` method is invoked and the values entered by the user are passed along so that the BPM system can use them (set them into a BPM Object for example). Then the instance moves forward in the process (unless you finish the `commitExternalActivity()` method with an Action different than OK or RELEASE).

- Configuration: you can optionally define an URL. When the task is executed from WorkSpace, WorkSpace redirects the execution to the URL.

## External Task Property Reference

Provides detailed information for External Task properties.

The following sections describes the properties defined in the Main Task window of an External Task.

### Properties

The following are available in the properties area. These properties are only available in Interactive Activity.

Property	Description
Repeatable	If true, the user is allowed to run this task more than once
Mandatory	If true, the user must run this task to complete the activity
Read Only	If true, this task cannot modify any instance variable

### Methods

Property	Description
Prepare Method	Method executed by the client PAPI application to retrieve values from the process instance
Commit Method	Method executed by the client PAPI application to submit new values to the process instance

### Configuration

Property	Description
Use configuration	Select the <a href="#">External Resources</a> on page 198. Click on <b>Edit</b> to modify the selected external resource.

## Input Tasks

Input tasks allow you present a simple form to the end user.

### Input Property Reference

The following sections describes the properties defined in the Main Task window of a Task using the Input Implementation Type.

### Properties

Property	Description
Mandatory	If true, the user must run this task to complete the flow object
Read Only	If true, this task cannot modify any instance variable

### Input Dialog

Property	Description
Input Dialog Title	Title for the dialog presented to the user. Click on <b>Instance Variables</b> to use the value of variable as the dialog title

**Buttons**

Property	Description
Assign Selected Button To	From the drop-down, select the instance variable where you want to save the name of the button pressed by the user
Cancel Button Is	From the drop-down, select the button used to Cancel the dialog

**Display Tasks**

Display tasks allow you present a simple information dialog to the end user.

**Display Property Reference**

The following sections provide detailed information for Display Implementation Type properties

**Properties**

Property	Description
Mandatory	If true, the user must run this task to complete the flow object
Read Only	If true, this task cannot modify any instance variable

**Display Dialog**

Property	Description
Display Dialog Title	Title for the dialog presented to the user.
Value	Message to be displayed to the user

**Buttons**

Property	Description
Assign Selected Button To	From the drop-down, select the instance variable where you want to save the name of the button pressed by the user
Cancel Button Is	From the drop-down, select the button used to Cancel the dialog

**Decision Task**

Decision tasks allow you to present a decision for to the end user. This is equivalent to the use of Decision Activities.

**Decision Property Reference**

The following sections provide detailed information for Decision Implementation Type properties.

**Properties**

Property	Description
Mandatory	If true, the user must run this task to complete the flow object.
Read Only	If true, this task cannot modify any instance variable

Decision Dialog

Property	Description
Decision Dialog Title	Title for the dialog presented to the user. Click on <b>Instance Variables</b> to use the value of variable as the dialog title

Buttons

Property	Description
Assign Selected Button To	From the drop-down, select the instance variable where you want to save the name of the button pressed by the user
Cancel Button Is	From the drop-down, select the button used to Cancel the dialog

Transitions

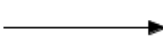

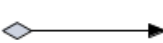



Transitions Overview

A transition advances the process from one flow object to another. In Business Process Modeling Notation (BPMN), transitions are also known as connecting objects.



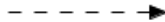
Transitions use directional arrows that display the direction of the flow. An instance flows through a process by following the logic that applies to a transition.

Transition Types

Oracle BPM provides many types of transitions. The most common transitions are: *Unconditional*, *Conditional*, *Due Exception* *Business Rule*.

Notation	Transition	Description
	Unconditional (Uncontrolled)	Instances flow through the transition without being affected by any conditions. In Oracle BPM, this is known as an uncontrolled flow.
	Unconditional (Default)	Instances flow through this transition when alternative condition transitions are not used. Oracle BPM automatically shows a default unconditional when at least one alternative condition flow is added to the flow object.
	Conditional	Instances flow through the transition if a specified condition is met.
	Business Rule	Instances flow through the transition if the specified dynamic business rule evaluates to true.
	Due (Timer)	Instances flow through the transition when a timer fires.
	Exception (Error)	Instances flow through the transition if an exception occurs.

The transitions in the following table—*compensate*, *message-based*, and *precedence*--are used less frequently. If you are just beginning to use Oracle BPM, you do not need to be familiar with these yet.

Notation	Transition	Description
	Compensate	Instances flow through the transition if compensation processing is required. The actions performed reverse (or undo) any work done in the previous flow object in the event that PBL-Method failure occurs.
	Message Based	Instances flow through the transition if a flow object that handles different argument sets receives a message. Available only from Begin or Message Wait events.
	Precedence	Only available in a Split-Join circuit. Copies within a Split-Join circuit can have a synchronization or a precedence. The precedence is represented by a dashed transition line and a solid arrowhead, not to be confused with the BPMN Message Flow, which begins with a circle and has an outline arrowhead.

### Which Transition Is Used?

All flow objects at least have an outgoing unconditional transition so there is always a way to continue the process. However, in most processes, *condition* transitions are also used. When one or more condition transitions originate from a flow object, the remaining unconditional transition is shown as a default flow transition.

In this case, the condition transitions are evaluated first, and the unconditional transition is taken only if the condition transitions all evaluate to `false`. In programming terms, the default unconditional is like the *else* clause in an if-then-else construct.



Business rule transitions are evaluated before condition transitions, so if a business rule transition and a condition transition both evaluate to `true`, the business rule transition prevails.

Due transitions act separately. They "pull" the instance from the flow object as soon as a timer fires. In this case, all other outgoing transitions are ignored.

## Adding a Transition

You add transitions in the process design editor. This task shows you how to add any type of transition using the process elements palette.

To add a transition:

1. Click on the **Transition** icon () from the palette, located in the **Flow** category.
2. Click on the flow object the transition will originate *from*.
3. Click on the flow object the transition will flow *to*.  
An unconditional transition is added to the process diagram.
4. To change this transition into another type of transition, right-click on it and click **Properties** ()

The **Transition** dialog box appears.

- a) Enter a name for the transition in the **Name** field.
- b) Click on the **Properties** tab.  
The **Properties** page appears.
- c) Select a transition type from the drop-down list in the **Type** section.  
A section showing properties corresponding to the transition you selected type will appear.
- d) Specify the data or options as required, and click OK.  
The transition is changed.

## Unconditional Transition

Provides detailed information on the Unconditional Transition.

When your process requires unrestricted workflow between two flow objects, you should add an Unconditional Transition. This type of transition indicates that no conditions exist to prevent instances from moving to the next flow object. Therefore, the transition occurs unconditionally.

After you create the transition, a line with a directional arrow connects the two flow objects on the design workspace. No icon is displayed next to the transition.



**Note:** If you do not want to see the Unconditional Transitions, disable the Show Unconditional Transitions property from the View menu, Transitions option.

### Rules

The following rules apply to unconditional transitions:

- Each flow object must have at least one **outgoing unconditional transition**. Exceptions to this rule are the following:
  - if the flow object has a **due transition**. The **Split and Multiple gateways** are the only ones that **must** have an unconditional transition, as an exception to this rule.
  - Global Creation, Global, Global Automatic and End flow objects have no outgoing transitions.
- Each flow object must have only one outgoing unconditional transition. Exceptions to this rule are:
  - Split gateways can have more than one outgoing unconditional transition.
  - Interactive activities may have more than one outgoing unconditional transition if the **User selects transition** check box is selected on the Interactive's **Activity Property** dialog box.
- Flow objects cannot have an unconditional and a conditional transition to the same destination flow object.

### Adding a Unconditional Transition

You can add unconditional transitions directly from the flow object context menu.

To add a unconditional transition:

- In the process design editor, right-click on the flow object *from* which the transition will flow, and click **Add unconditional transition**.
- Click on the flow object the transition will flow *to*. Note that as you move the mouse, the transition line is shown.

The unconditional transition is added to the process diagram.

## Conditional Transition

Provides detailed information on the Conditional Transition.

When your process requires restricted workflow between two flow objects, you should add a **Conditional transition**. A Conditional transition indicates that workflow will only occur if specified conditions are met. The special conditions are added by using a PBL-Method in the **Condition** field in the **Transition Properties** dialog box.



For example, in an Order Management process, a conditional transition directs instances from the **Review Order** activity to the **Special Care** activity if the order status is equal to "Expedite" or "Alert". The PBL-Method for this condition is as follows:

```
orderStatus in ["Expedite", "Alert"]
```

After you create the transition, a line with a directional arrow connects the two flow objects on the design workspace. The icon next to the transition indicates that it is a **Conditional transition**.

As well, the transition's **Name**, **Description** or **Condition** may appear next to it depending on the chosen preference.



**Note:** If you do not want to see the **Conditional transitions**, disable the **Show Conditional Transition** property from the **View** menu, **Transitions** option.

## Rules

The rules for using conditional transitions are as follows:

- Conditional transitions are available for most flow objects, with the exception of End and Multiple. (Global, Global Creation and Global Automatic do not require transitions.)
- Flow objects cannot have an unconditional and a conditional transition to the same destination flow object at the same time.

## Defining the Condition

The **Condition** is defined in the Transition Properties dialog box by typing a PBL-Method in the **Condition** field. By default the transition's **name** is used as the expected **result** of the condition.

For example if the transition represents the normal flow then you can **name** it as **OK** and the condition is automatically built as **result == "OK"**. This is the default condition and is valid while the condition is empty.

If you manually define a condition then the default condition is no longer valid.

Instance variables can be used in the condition as well.

More than one conditional transition might be required. Multiple conditional transitions flowing out of a flow object are ranked in order to determine the evaluation precedence. The precedence is assigned in ascending order according to the creation order given when the process is first designed in Studio. Nevertheless, the conditional transitions' order can be changed by using the **Conditional transitions order** properties window that Studio displays in the flow object shortcut menu (right-click on the flow object) when more than one conditional transition has been defined.



**Note:** it is highly recommended to **name** the condition to represent its condition. Furthermore, use the predefined variable **result** to set the result of it.

- The syntax is automatically checked. If everything is correct, a **blue flag** appears on the upper-right corner of the **Conditional** field. If something is incorrect, a **red flag** appears. Drag the mouse over the red line below the red flag and the error displays. In addition, the error is shown at the bottom of the dialog box if you click on the statement that has the problem.
- Click **OK** to close the Transition Properties dialog box. When the dialog box closes, the last check is performed. If something was not corrected, the error message will display.



**Note:** It is not recommended to use a variable type ANY because, in order to compare it, you will have to cast it before. If this is not done, the comparison might fail.

See Process Business Language Programming Guide for further information.

## Adding a Conditional Transition

You can add conditional transitions directly from the flow object context menu.

To add a conditional transition:

1. In the process design editor, right-click on the flow object *from* which the transition will flow, and click **Add conditional transition** (?).
2. Click on the flow object the transition will flow *to*. Note that as you move the mouse, the transition line is shown.  
The **Transition** dialog box appears.
3. In the **Name** field, enter a name for the new transition.
4. Switch to the **Properties** page of the **Transition** dialog box.
5. Enter a conditional expression using PBL syntax. This expression should evaluate to a boolean (a value which is either `true` or `false`).

The condition can use instance variables. To see the instance variables which are available, click on the **Instance Variables** icon.

The following illustrates valid and invalid conditional expressions, where `total` is an instance variable of type `Decimal`:

Valid Expression	Invalid Expression
<code>true</code>	<code>"true"</code>
<code>total &gt; 2500</code>	<code>2500</code>
<code>total &gt; 0</code>	<code>total</code>

6. Click **OK**.

The conditional transition is added to the process diagram.

## Business Rule Transitions


Business rule transitions are a special kind of conditional transition which evaluates a dynamic business rule instead of an expression. Business rule transitions are evaluated before conditional transitions.

### Adding a Business Rule Transition

You can add business rule transitions directly from the flow object context menu in the process design editor.

To add a business rule transition:

1. In the process design editor, right-click on the flow object *from* which the business rule transition will flow, and click **Add business rule transition** (\$).
2. Click on the flow object the transition will flow *to*. Note that as you do move the mouse, the transition line is shown.  
The **Transition** dialog box appears.
3. In the **Name** field, enter a name for the business rule transition.
4. Switch to the **Properties** page of the **Transition** dialog box.
5. In the **Select Business Rule** section, select a business rule from the drop-down list.

 **Note:** If you have not yet defined any business rules in the project, click **New** and give the business rule a name. After completing this task, you must edit the business rule following the steps in [Defining a Business Rule](#) on page 228.


6. Click **OK**.

The business rule transition is added to the process diagram.

## Due Transition

Provides detailed information on the Due Transition.


A Due Transition is used when a process requires an instance to move to the next flow object within a specified time period. Due transitions are used to implement deadline processing.

 **Note:** Flow objects can use only one due transition to link to another activity.

After you create the transition, a line with a directional arrow connects the two flow objects on the design workspace.

The icon next to the transition indicates that it is an **Interval expression or Interval constant due transition**. The due condition is displayed as well.

The icon next to the transition indicates that it is a **Scheduled based due transition**.

 **Note:** If you do not want to see the Due Transitions disable the **Show Due Transitions** property from the **View** menu, **Transitions** option.

## Defining the Due Condition

The Due Condition can be defined as Schedule based, Interval expression, or Interval constant.

In any of these cases, you can decide to use Calendar Rules. So, if the calculated due date is not a working day, the applicable due date is moved to a valid date.

### Schedule based

If the next scheduled due time falls on a non-working day, you have the ability to select whether the due time is passed to the **Next working day, same time** or **Next scheduled based time**.

For example, if you set the due time to happen once a week, every Monday at 11:00 AM and one Monday is a holiday, then if you have selected:

**Next working day, same time** the due time is set to Tuesday at 11:00 AM, or


**Next scheduled based time** the due time is set to next Monday at 11:00 AM.

The Scheduled option indicates that the due transition is set to run on a specific date at a specific time.

The **DAILY** choice allows you to select a time you want the due time to apply each day. For example, every day at 3:00 AM.

The **WEEKLY** choice allows you to select a "**day of the week**" and a time that the due time applies every week. For example, every **Monday** at "**10:18 A.M.**"

The **MONTHLY** choice allows you to select the month, the week of the month (First-Second/Fourth-Last)/the day of the week (Sun-Sat) or the day of the month and the day (1-31) and the time to which the due time applies. For example, you can choose the third (3rd) Thursday of every month at 3:45 P.M.

 **Note:** Notice that the due is moved based on the day and not on the time as explained on the dialog box

### Interval Expression

The Interval Expression option allows the due time to begin the moment the instance arrives at the flow object plus an interval time. The interval time to add to the arriving time can be the result of an expression.

If the calendar rules apply and the calculated due time is on a non-working day, the due time is postponed to the first working time.

- In the Due Interval field, type a time interval, making sure that you surround the time interval with a single quote. (See below for time syntax examples.) For example, if you require a time interval of five minutes, you need to type '5m'. This means that the instance has five minutes (5m) to flow through the Due transition from the source flow object to the next flow object. To implement this example, you can also define the Due transition as an Interval constant (see below for more information).

If you require a more complex condition, add a PBL-Method in the Due Interval field to force an instance through the process in a specified amount of time under certain conditions. In the following example, the PBL-Method indicates that an order greater than \$5000 can sit in the Account Manager's queue for only 2 minutes ('2m'). If the order is less than \$5000, the order can remain in the queue for 5 hours ('5h').

```
(OrderAmount > 5000) ? '2m' : '5h'
```

- The syntax is automatically checked. If everything is correct, a green flag appears on the top right corner of the Conditional field. If something is wrong, a red flag appears. The error is shown at the bottom of the dialog box if you click on the statement that contains the problem.



**Note:** Time will start running immediately after the instance reaches the source flow object of the due transition.

### Time syntax

Due interval logic to indicate an interval of time is added to the due transition properties. The syntax is:

- d for day
- h for hour
- m for minute
- s for seconds

Examples of valid intervals to enter in Due transition logic include:

- '3d' for three days
- '1h' for one hour
- '4m' for four minutes
- '2m30s' for two minutes and 30 seconds.

See METHODS REFERENCE GUIDE for further information on time and interval syntax.

### Interval Constant

The Interval Constant option allows the due time to begin the moment the instance arrives at the flow object plus a defined time (number of months/days/hours/minutes/seconds).

If calendar rules apply and the calculated due time is on a non-working day, the due time is postponed to the first working time.

For example, you decide that the due time is always the instance arrival time plus 7 days and 12 hours.

No expressions are available. If so, define the due time as an Interval expression.



**Note:** Time will start running immediately after the instance reaches the source flow object of the due transition.

### Priorities for the Due times

A due transition can be specified for leaf flow objects as well as for group activities. Therefore, an instance can be affected by multiple due times as follows:

- the flow object due transition interval,

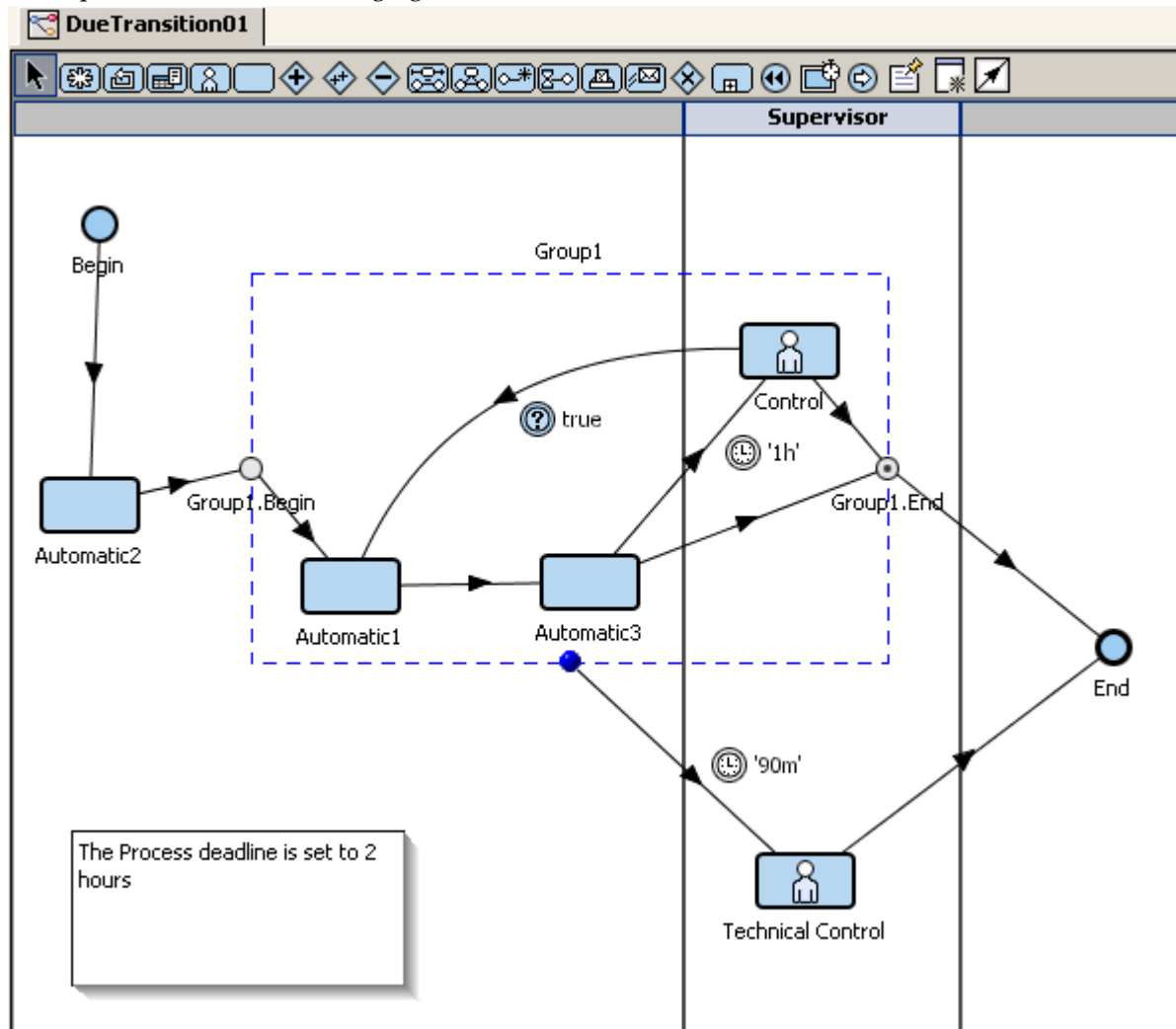
- each nesting group due transition interval,
- the process instance deadline.

When an instance arrives at a flow Object, these three options are considered in order to determine the shortest due time and which due transition is first priority.

You should consider that a group's due transition time begins when the instance arrives at the first flow object within that group.

The due time assigned to an instance, as it arrives at a flow object, is the shortest time of all the three possible due time options listed above.

For example, as shown in following figure:



**Figure 1: Due Transition Example**

when the instance arrives at the activity **Automatic3**, as this activity has a due transition, the due time is the shortest time between:

- 1 hour: defined in the Automatic3 due transition.
- The remaining time of 90 minutes as defined for the Group1 due transition. This means that if the instance has remained for 50 minutes in the Automatic1 activity, the remaining time for the group is 40 minutes. In consequence, the instance will probably flow through the Group1 due transition to the Technical Control activity instead of flowing to the Control activity (1 hour).
- The remaining time of 2 hours defined as the process due deadline. If the instance has been in the Automatic2 Activity for 100 minutes, the remaining time for the process is 20 minutes. Thus, the instance

will probably flow to the End Event instead of flowing to the Technical Control activity (90 minutes) or to the Control activity (1 hour).

If more than one due transition expires at the exact same time, the instance is sent through the outermost due transition (of those transitions which time interval expired at that moment).

For example, the instance is within a flow object that has a due transition that expires at noon. Additionally, this flow object belongs to a group that has also a due transition that expires at noon. At noon the instance will flow through the group due transition.

When a due interval is calculated for a flow object (leaf or group), the calendar rules are taken into account as regards the role where the instance is or whether the group spreads over many roles.

### Task Timeout

If a task is running, the time it has to complete depends on the task timeout, defined with the predefined variable `timeout` or 5 minutes by default. But the due time explained above also applies. Therefore, if there is a task running for a specific instance and any of the due time at flow object, group or process level expires, although the task might have some remaining time, the task will finish and the instance will flow through the corresponding due transition.

Moreover, if the task timeout was set with an interval greater than specified as the Engine property "Maximum task timeout", the task will fail until the process or maximum task timeout is fixed.

Continuing with the example image above, the Automatic1 activity has a task that has set a timeout to 30 minutes. And the instance has been in the Automatic2 activity for 100 minutes, the remaining time for the process is 20 minutes. So, although the task theoretically has 30 minutes to execute, after 20 minutes the process due deadline expires, the task is terminated and the instance flows to the End event.

### Due Time Calculation Failure


If a due time calculation fails, the transaction of the flow object from which the instance came from, fails.

For example, if you send an instance from an Interactive Activity to an Automatic Activity that has a due transition but its calculation fails, the previous **Interactive** activity execution fails.

### Adding a Due Transition

You can add due transitions directly from the flow object context menu in the process design editor.

To add a due transition:


1. In the process design editor, right-click on the flow object *from* which the transition will flow, and click **Add due transition** .
2. Click on the flow object the transition will flow *to*. Note that as you move the mouse, the transition line is shown.  
The **Transition** dialog box appears.
3. In the **Name** field, enter a name for the new transition.
4. Switch to the **Properties** page of the **Transition** dialog box.
5. Select an expression type, which can be *Schedule Based*, an *Interval Expression*, or an *Interval Constant*.
6. Enter the data required and click **OK**.

The due transition is added to the process diagram.

## Exception Transition

Provides detailed information on the Exception Transition.

An Exception Transition is used to submit the instance to an Exception Handler flow when the flow object or group of activities fail or throw an exception.

 **Note:** Flow objects or groups can use only one exception transition to link to the exception flow. The exception flow can have as many flow objects as required to fix the exception condition.

After you create the transition, a line with a directional arrow connects the two flow objects on the design workspace.

The source of the transition is the flow object or group where the exception occurred, and the target is the first flow object in the exception handler flow. As the exception handler flow is independent from the main process flow, it cannot have transitions back to the main process flow.

**Exception holder variable:** you can create and define an instance variable as the **Exception holder variable**. If you do not define one, Studio creates it automatically.

Once the exception occurs it is stored in this variable and it is available within the exception flow that is handling the exception. It can be used by the developer to debug or analyze the exception in depth.

You can create only one variable and re-use it in all the exception transitions or you can associate different instance variables to each exception flow. Normally the variable type matches the exception type.

If there is no variable defined (backward compatibility), a default exceptionHandler instance variable (Any type) is created when you check the design.


At runtime if the variable receives an unmatching type, then a warning is logged.

The variable content is available only within the Exception flow to where the instance is routed through the exception transition. Therefore the exception contained in the variable is not propagated to other processes.

## Compensate Transition

Provides detailed information on the Compensate Transition.

A Compensate Transition is used when an activity or group of activities require that the actions performed by the BP-methods should be reversed. Reversal is needed in case of total or partial BP-method failure, which could be caused by any number of things such as a call to an external system that fails, equipment failure, a database call with bad data and so on.

 **Note:** Activities or groups can use only one compensate transition to link to the compensate flow. The compensate flow can have as many activities as required. However, no other compensate transition between them applies.

After you create the transition, a line with a directional arrow connects the two activities on the design workspace.

The source of the transition is the activity to be compensated and the target is the first activity in the compensation handler flow. Since the compensation handler flow is independent from the main process flow, it cannot have transitions back into the main process.

For further information, see Compensate Handling and Compensate Activity.

## Message Based Transitions

Provides detailed information on the Message Based Transition

Message Based Transitions are available for the Begin and Message Wait event types. A Message Based Transition can be added by using the source flow object argument mapping sets.

Begin and Message Wait events can receive different sets of arguments. Each set has a name defined as the Argument set name. Within each set, the arguments can be mapped to instance variables or predefined variables. For any of these mappings, a new outgoing transition can be added to the flow object. The transition type is called Message Based Transition. Basically, you define the transition that the instance will flow through based on the received message.

There cannot be more than one outgoing message based transition for the same Argument set name.



## Variables

Variables are placeholders for values in your process. Each variable has a name, description, type and value. Oracle BPM provides different classes of variables based on the scope and context where they are used.

### Order of Precedence within a PBL method

Variable names are resolved in the following order of precedence:

1. Local
2. Argument
3. Instance

When accessing variables within a method, you should be aware of the order of precedence among different types of variables. This is important if you have variables of different types that have identical names.

To explicitly state the scope of a variable, use the following prefix keywords:

Keyword	Description
this.	Explicitly specifies an instance variable, including project and predefined variables.
arg.	Explicitly specifies an argument variable.

## Creating Project and Instance Variables

Outlines procedures for creating and editing Variables.

Before beginning this task, ensure that the Variables view is visible.

Instance and Project Variables are created and edited within the Variable view. You can also use the Variables view to map incoming and outgoing Argument Variables.

1. Click the + icon in either the Project or Instance Variable area, depending on which type of variable you are creating.  
The Variable properties window appears.
2. Enter the Name and Label of the Variable.
3. Select the Variable type.
4. If you are creating a String or Decimal Variable, specify the variable size.
5. Click **OK**.

The new Variable appears in the Variables view.

## Instance Variables

Instance Variables are custom variables whose scope is contained within a process. Instance Variables contain information that flows through the process from the **Begin** to the **End** activities. The value of each variable is stored independently for each process instance.

Most activities within a process can access and modify the value of an Instance Variable, with the exception of Global Creation and Global Automatic activities. Global Interactive activities can access instance variables only when the **Has Instance Access** property of the activity is checked.



Examples of instance variables that may be found in a shipping order management process include:

invoiceNumber, customerName, customerNumber, orderStatus, orderAmt and shipStatus.



## Instance Variable Storage Categories

Instance variables have a property called *category*. This property determines how the Process Engine stores the value of the variable in the database, but does not affect the logic of the process. This property takes one of the following values:

Category	Description
Normal	<p>By default instance variables are defined as <i>normal</i>, and only those that have some special characteristics need to be categorized in a different way. For each process instance, the process execution engine stores all normal instance variables together by serializing their values into a single BLOB in the Process Execution Engine database. After the execution of each process activity, the Engine updates this BLOB in the database.</p> <p>To keep resources under control, the process engine limits the size of this BLOB for each process instance. The <b>Maximum Instance Size</b> property of the Engine defines this limit (in kilobytes), and is configurable from the Process Administrator application. The default value is 16 KB.</p>
Separated	<p>The process execution engine stores separated variables in a separate table in the database.</p> <p>The Separated storage category is intended for variables which must hold large amounts of data (10 KB or more) and are used (and modified) rarely in the process.</p> <p>A common use case for separated variables is when the input to the process is a potentially big XML document, which must be parsed only once and is not modified throughout the process. If the variable holding the XML is defined as Normal, every time the process instance flows from activity to activity the XML is serialized to the database along with the other Normal variables (even if the XML was not modified). By defining the variable as Separated, the Process Engine updates Normal variables independently, updating the XML only if it was modified.</p> <p> <b>Important:</b> In Multiple Gateways, separated instance variables values are not automatically copied to the separated instance variables of the copies. They have to be copied manually in the Multiple Gateway PBL method.</p> <p> <b>Note:</b> The size of a the BLOB for Separated variables is only limited by the underlying DBMS used by the Process Execution Engine. Mind though that storing big data elements as instance variables (Normal or Separated) is not recommended as it has a negative impact on Engine performance.</p>

## Default Values

Instance Variables are initialized based on their type according to the following table:

Type	Default Value
Numeric (int, real, decimal)	0
Boolean	FALSE
All other types	Null

## Accessing Instance Variables

All PBL methods of the process can refer to any instance variable by the variable's name. You may also use the explicit `this.` prefix to avoid naming conflicts with variables defined at another scope (i.e.: argument or local variables).

## Predefined Variables

Predefined variables are special instance variables that are always defined for all BPM processes. All processes include these predefined variables and cannot be removed. The value of some predefined variables cannot be modified by the process logic.

Unlike with regular instance variables, end users can search for process instances using predefined variables in their search conditions.


### Predefined Variable Reference

The following table lists the Predefined Variables:

Predefined Variable	Permitted Values	Type	Description
action	Modifiable : OK ; FAIL ; RELEASE ; CANCEL ; REPEAT ; ABORT ; BACK ; SKIP ; NONE	Fuego.Lib.Action	The value of the <code>action</code> defines the outcome of the current activity transaction. Depending on this value, the Process Execution Engine decides what to do with the current process instance after the execution of the activity is finished. The <code>action</code> variable is reset before every transaction. See <a href="#">Action Variable</a> on page 126 for details.
activity	Read only	Fuego.Lib.Activity	Holds an object representing the current process activity.
activity.deadline	Read only	Time	Set automatically when there is an outgoing Due transition on the current activity (receptionTime + the due transition time interval.)
activity.source	Read only	Fuego.Lib.Activity	The activity from which the current instance came into the current activity. Null in the case of Global and Begin activities. When the process instance is in an exception handling flow (or Termination Wait flow), the value of <code>activity.source</code> refers to the activity that raised the exception (or received the notification).

Predefined Variable	Permitted Values	Type	Description
attachments	Read only	Fuego.Lib.Attachment[]	Array holding the process instance attachments.
children	Read only	String[ordered Object]	Array of the Ids of the process instances that are children of this instance, ordered by the list of activities that created them. Children instances are those instances created by the <b>Subflow</b> activity or <b>Process Creation</b> activity.
creation.participant	Read only.	Participant	Participant (end user) that created this process instance.
creation.time	Read only	Time	Creation time of this process instance.
currentException	Read only	String	When the process instance is within an exception handling flow, this variable holds the name of the exception.
deadline	Modifiable.	Time	The Process Execution Engine raises an <del>InstanceExpirationException</del> on the process instance if the instance does not reach the <b>End</b> activity before the time specified by deadline.
description	Modifiable	String	Descriptive name of the instance that appears in WorkSpace. By default: ProcessName + id.number.
id.id	Read only	String	The process instance unique identifier. It includes the deployed process name, organization, organizational unit and the instance Id (including thread id).
id.number	Read only	Int	A number that uniquely identifies an instance within an Engine.
id.copy	Read Only	Int	The current instance copy (thread) number. Copies are those instances created

Predefined Variable	Permitted Values	Type	Description
			by a Multiple Gateways and Split Gateways.
notes	Read only	Int	An array of all notes added to an instance.
organization	Read Only	String	Name of organization where the process is running.
organizationalUnit	Read Only	String	Name of the process Organizational Unit, such as Marketing or Finance.
parent.id	Read only	String	The Id of the parent instance of the current instance if there is one; null otherwise. In the case of a Procedure, it contains the id of the calling process
parent.copy	Read only	Int	The parent instance thread number.
parent.number	Read only	Int	A number that uniquely identifies the parent instance within an Engine.
participant	Read Only. Any participant	Fuego.Lib.Participant	The human participant (end user) that is currently processing the instance.
participant.locale	Read Only	Fuego.Util.Locale	The language, country and variant, if applicable, of the current participant.
participant.next	Modifiable: Any participant	Fuego.Lib.Participant	The participant to which the instance will be assigned when it arrives to the next activity.
participant.sticky	Modifiable: Bool	Fuego.Lib.Participant	If <code>participant.sticky</code> is set to true, variable <code>participant.next</code> is automatically set to the current participant (that is, the value of <code>participant</code> ). Each time the instance moves to a new activity, it is assigned to this participant (provided the participant has visibility over the activity).
priority	Modifiable: 1 Lowest ; 2 Low ; 3 Normal ; 4 High ; 5 Highest	Int	Priority of the instance. This variable is intended to be used as a hint to

Predefined Variable	Permitted Values	Type	Description
			participants regarding the urgency of each work item. The value of <code>priority</code> has no effect on how the Process Execution Engine handles each process instance.
<code>process</code>	Read Only	<code>Fuego.Lib.Process</code>	The process the instance belongs to.
<code>process.id</code>	Read Only	String	The Identifier of the deployed process containing the deployed process name, its organization and organizational unit.
<code>process.idNumber</code>	Read Only	Int	A number that identifies the process inside the Engine
<code>process.name</code>	Read only	String	The name of the current process.
<code>receptionTime</code>	Read only	Time	The time when the current instance arrived at the current activity.
<code>result</code>	Modifiable: Any string	String	<p>Generic String you can use to set to indicate the outcome of the current activity. Intended to be used for evaluating outgoing conditional transitions. The value of <code>result</code> is reset when the instances arrives to a new activity.</p> <p> <b>Important:</b> For backward compatibility, if the action variable is not set in a PBL-method and the value of <code>result</code> is one of the possible values of <code>action</code>, the Engine maps the value of <code>result</code> to variable <code>action</code>. For example, if variable <code>result</code> is set to "FAIL", this is equivalent to setting</p>


Predefined Variable	Permitted Values	Type	Description
			action to Action.FAIL.
status	Read Only. RUNNING, EXCEPTION, SUSPENDED, GRABBED, COMPLETED, ABORTED, ACTIVITY_COMPLETED	ProcessInstanceState	Current status of the process instance.
timeout	Modifiable	Interval	The amount of time that a PBL-Method transaction has to complete before the Process Execution Engine aborts its execution. By default this variable is set to 5 minutes. The <b>Maximum Timeout</b> property of the Engine defines the maximum value you can assign to variable timeout.
totalCopies	Read only	Int	Number of copies of an instance.

### Action Variable

When a PBL-Method is executed, multiple outcomes can result from the execution. PBL-Method execution status is indicated by the value of the predefined `action` variable.

Depending on the value of `action`, the Engine responds accordingly and saves or undoes (commit or rollback) the changes made to any variable when the PBL-Method is executed. The following table lists the valid values for the Action Variable:

action =	Description	Result
OK NONE	Indicates that PBL-Method execution was successful. This is the default value.	PBL-Method changes to "completed" status. If the activity is marked as auto-complete, the instance flows to the next activity according to transition rules.
FAIL	Indicates that the PBL-Method has failed its execution. The PBL-Method must be executed again.	If a rollback PBL-Method is included, it is executed. The Engine retries the original PBL-Method until it succeeds or reaches the maximum number of retry times. In the latter case, the instance is routed to an Exception handling activity. The effect of FAIL is equivalent to that of a <a href="#">System Exceptions</a> on page 231.
CANCEL	PBL-Method execution is aborted.	The instance state is rolled back to the point before PBL-Method execution. No trace of the

action =	Description	Result
		PBL-Method failure or execution appears in the audit trail. CANCEL is only relevant on interactive executions.
REPEAT	Indicates that the PBL-Method execution is successful, but not recorded as completed.	REPEAT is only relevant on interactive executions. The transaction is committed. However, the task's status remains in a pending state and the task should be executed again. REPEAT indicates that, although the task was successfully executed, it remains <b>pending</b> . Therefore, if the task is mandatory, the participant has to execute it again.
RELEASE	Ends the PBL-Method execution and releases the instance from this activity.	The transaction is committed. The instance is released to the next activity without processing any of the PBL-Methods in the current activity, even if they are marked mandatory. RELEASE is only relevant on interactive executions and on Join activities.
ABORT	Ends PBL-Method execution and aborts the entire process instance.	The instance is not processed and is sent directly to the End activity.  <b>Caution:</b> Instances that are aborted cannot be recovered.
BACK	Ends PBL-Method execution and sends the instance back to the activity where the exception (or interruption) occurred.	The transaction is committed. Used in an exception handling flow (or notification flow) to send the instance back to the activity where the exception (or interruption) occurred.
SKIP	Ends PBL-Method execution and sends the instance back to the activity where the exception (or interruption) occurred and skips it.	The transaction is committed. Used in an exception handling flow (or notification flow) to send an instance back to an activity in the process and skips its execution. The instance goes back to originating activity and is released to the next activity, without re-executing the activity that caused the failure.

#### Action Variable Example

The following PBL-Method example shows how you can manually set the `action` variable if a Boolean expression evaluates to **true**.

```
if selectedButton == "Yes" then
  action = OK
```

```
elseif selectedButton == "Abort" then
    action = ABORT

else
    action = BACK
```

## Project Variables

Project variables are special instance variables that are declared at the project level and appear in all processes of the project. Unlike regular instance variables, end users can see project variables in work list columns of the WorkSpace and use project variables in search conditions.

Like regular instance variables, the value of each project variable is stored independently for each process instance.

Project instance variables have special advantages and limitations in comparison to regular instance variables:

- You define a project variable once for a given project. The characteristics of a project variable, such as name, type, or length are the same for all processes in the project.
- You can only define Project Variables of basic types `String`, `Int`, `Bool`, `Real`, `Decimal` and `Time`.
- End users can view project variables in the WorkSpace **Work List** panel and sort the lists by project variable.
- End users can search and filter process instance based on the value of project variables.

### Project Variable Storage

The Process Execution Engine stores project variables into their columns in the database.



**Note:** Each project variable adds a new column to the Engine database. If multiple projects are deployed to the same Process Execution Engine, the same column can be reused for variables of different projects. You cannot have more than 256 project variables in a single project.

### Defining Project Variables as a Business Indicator

You can define a Project Variable as a Business Indicator. A Business Indicator is used primarily to generate Business Activity Monitoring (BAM) and Business Activity Data Mart information.

## Local Variables

Local Variables store information that is used only inside the scope of a single PBL method. The scope and lifetime of a local variable is within method itself. Once the method has been executed, the information stored in a local variable is lost.

## Screen Flows

### Screenflow Overview

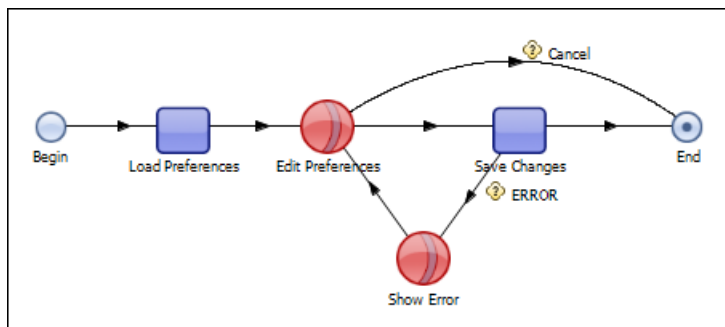
A screenflow is a user interaction flow. Screenflows are similar to processes in that they are designed graphically, have a start and an end activities, support conditional expressions, and have their own instance variables.

Screenflows differ from business processes in that the entire sequence of a screenflow is executed by a single participant, so the first thing you will notice in a screenflow design editor is that there are no swimlanes.



Screenflows are called from an interactive activity. The state of a screenflow is not persisted in the process instance till the screenflow is complete and returns to the calling activity. If a participant starts to execute a three page screenflow and inputs information into the first two pages and then logs out, it is as if he had input nothing.

You can call a screenflow are called from any interactive activity of any process in the project. Use them to implement interactive tasks.



**Figure 2: Simple Screenflow example. Note that there are no swimlanes.**

Screenflows should be thought of as components with a process-like interface. Since screenflows are not actual business processes, they are built from a much smaller set of components. Only the following activities are supported:

- Begin
- End
- Interactive Component Call
- Automatic
- Subscreenflow

These activities have similar behavior as those in the process, but they are much simpler. In addition, screenflows support three types of transitions:

- Unconditional
- Conditional
- Exception
- Due

You should see interactive activities as a point in the process where instances get into an inbox for something to be done. The execution of this activity task will involve the invocation of interactive components. This sequence of interactive component invocations should be mapped to the screenflow's activities. Later on, this screenflow should be mapped as the implementation of the interactive activity.

### When to Use Screenflows

You should use screenflows in most interactive tasks whenever possible. They have several advantages over BPM Objects with hand written interactions:

- Virtually no code is necessary to connect the different presentations.
- The flow between the screens is explicit and graphically shown.

### Differences between Screenflows and Sub-processes

The most important differences are the following:

- Screenflows are executed by a single participant to complete a process task.
- They are designed to be used as a single task within a process.

### Model-View-Controller Analogy

If you are familiar with Model-View-Controller (MVC) architecture, you can think of a screenflow as the controller, while the BPM Object is the model, and presentations are the view. Just as with an MVC pattern, the screenflow, acting as the controller, determines which page the user will see, and is responsible for initializing it to whatever values are required.

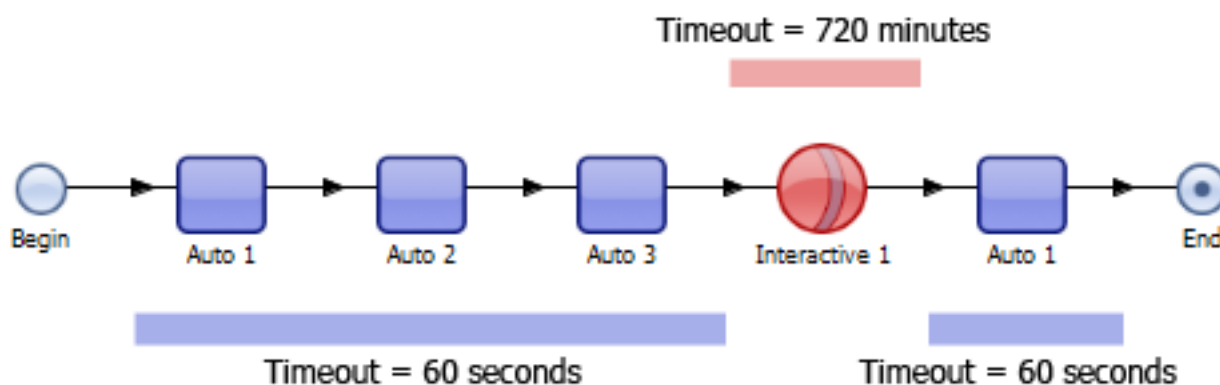
Screenflow logic also handles user responses. For instance, note that in the example screenflow shown above, a conditional transition is taken when the user clicks on the Cancel button in the Edit Preferences Interactive Component Call. There are some characteristics, which do not fit exactly with the Model-View-Controller model. For example, presentations are a part of the BPM Object definition, while an ASPX or JSP page is an independent file.

### Screenflow Timeout

A timeout defines the amount of time a task will wait to complete an action before processing continues.

If you deploy a process that has a screenflow, you need to consider that some timeout settings are available in the Process Administrator. You can set the PBL-Method timeout and the Interactive component timeout

In a screenflow you can combine different types of activities and timeouts apply for each activity or group of them. In the following example the PBL-Method timeout is set to 60 seconds and the Interactive component timeout is set to 720 minutes.



**Figure 3: Screenflow Timeout Example**

Once the screenflow begins to execute, the first 3 automatic activities (Auto 1, Auto 2, and Auto3), will have a *total* timeout of 60 seconds. When an interactive activity is reached, the Maximum PBL-Methods timeout is reset.

Then the Interactive Component Call activity, Interactive 1, begins to execute. The user has 720 minutes maximum to complete the task.

To complete the screenflow, the Auto 4 activity executes and has 60 seconds to complete its execution.

## Procedures

### Procedures Overview

Procedures provide a graphic definition of component methods that do not have interaction with end users. Procedure use a graphical syntax similar to Oracle BPM Studio processes.

Procedures contain a set of activities that is executed by a single participant. It also has a reduced set of activities that can be used. No roles are allowed because a procedure is limited to automatic activities.

Procedures are designed to be used in any part of a process. A procedure cannot be used outside the project that it belongs to, but it can be reused among Processes in the same Project.

As it has an automatic behavior, a procedure does not have roles and it only includes automatic activities. That is, activities of the following types: Begin, End, Process Creation with no Termination Wait activity, Process Notification, Automatic, and Split-Join). It also includes Groups and Compensate Transitions.

Automatic Activities within the procedure can have:

- Process Business Language (PBL) Methods
- Component calls (Runs on server components only.)
- Procedure calls.

## When to Use Procedures

Procedures should be used in order to reuse part of a process. They are the right way to share process behavior between more than one process or inside the same process (calling the same procedure several times) instead of using IPC.

A procedure is a set of automatic activities that does not define a Process, but it defines some automatic instance behavior. For example:

- Having a set of automatic activities that perform several notifications to different third-party applications and this behavior is repeated by most of the processes within the project.
- A set of automatic activities that should be used several times within a Process. Therefore, adding several Procedure calls to the same Procedure would clarify the design.
- If your PBL-Method has a large number of lines, you should consider the possibility of moving the PBL-Method to be implemented as a procedure and break down the method into several activities in a graphical design.
- To graphically show a sequence of components methods to improve its comprehension.
- To hide certain details of the implementation and push some logic to another level of the design.

## Procedures within a Process Instance

Once the process instance reaches an activity that executes a procedure, it remains there until the procedure finishes. The process instance is never in the procedure. The process instance actually remains in the process therefore, whatever you 'apply' in the procedure does not affect the process instance.

The procedure instance is a separate instance and the operations performed over the instance within the procedure will not apply to the process instance. For example, neither adding notes nor attachments will apply to the process instance. If you need to use any of the process instance variables data, they need to be passed as arguments.

## Rolling Back Procedures

Procedures are defined as atomic. The rollback is automatically done by the Engine.

Operations guaranteed to be rolled back are:

- Transactions performed using External Resources that handle transactions such as Data Base transactions or EJB. In a BPM J2EE Engine the rollback is guaranteed by the two-face commit. In a BPM Engine Standalone, the commit is an optimist commit. This means that each external resource receives an independent commit but it is expected that neither of them will fail, but if one does, then the transaction is not fully rolled back. For example, if you have two databases with different transactions, and the second one fails, the commit performed to the first one is not rolled back.
- Update to Instance information.
- Update to variables information.

Exceptions cannot be treated inside a Procedure, they are thrown to the immediate outer group (or parent group).

### Creating a Procedure

The following steps describe how to create a new Procedure:

- 1. Right-click on the Processes resource in the **Project Navigator**.
- 2. Select **New Procedure**.
- 3. Enter a Name for the new Procedure and an optional description.
- 4. Click OK.

The new Procedure opens in an editor window showing the default Begin and End Activities.

- 5. Add required activities to the procedure.

Procedures are available under the Process resource in the Project Navigator.

### Organizations




This section provides general information about organizations and provides procedures for creating and maintaining an organization using Oracle BPM Enterprise. It also provides information on creating and configuring directory services.





#### Organization Overview

Business processes that require user interaction generally occur within the context of an organization. Defining an organization allows users participate in your business process once it is published and deployed. It also ensures that users can only perform activities appropriate to their role within the organization. Each Oracle BPM Project must have an organization defined.

Within Oracle BPM an organization defines a hierarchical structure that reflects the real-world organization of your business. An Oracle BPM organization defines the way people are grouped and defines the roles or each group and individual.

The following table lists the elements of an Oracle BPM organization.

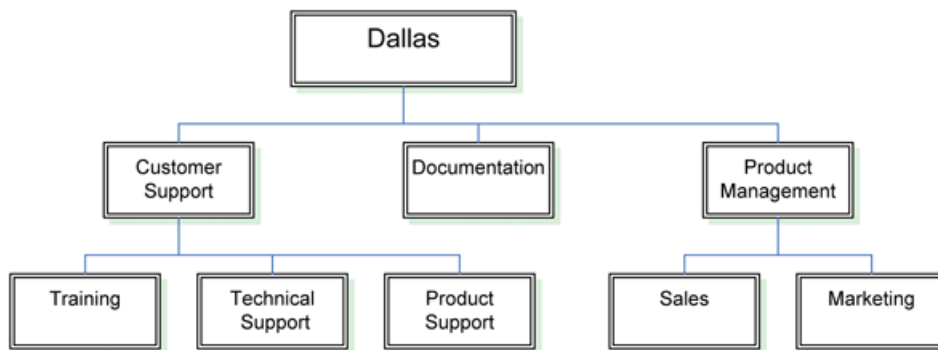
Element	Icon	Description
Organizational Units		Organizational Units are used to represent departments or divisions within the organization. Organizational Units can be defined hierarchically so that, for example, you can represent divisions within an organization, departments within a division, areas within a department, and so on. You can assign Participants, Calendars, and Business Parameters to an Organizational Unit. You can also deploy processes under an organizational unit.
Roles		Roles are used to represent functions performed by people related to the organization. Roles are assigned to participants or groups, and these assignments define the permissions the participants have when executing Oracle BPM tasks through WorkSpace.
Groups		Groups are collections of roles. In this way, it is possible assign multiple roles to participants in a single step. Groups may also contain other groups.

Element	Icon	Description
Participants		Participants are the actual people who participate in the organization, usually as end users of the BPM implementation.
Holidays		Holidays Define the organization's non-working days. These rules inform the Process Execution Engine that there is an exception to the normal calendar rules on certain days of the year.
Calendars		Calendars define the organization's work week and work schedule. Calendar rules can be assigned to organizational units.
Business Parameters		Business Parameters are used to maintain constant values defined either for the entire organization, or at the Organizational Unit level. These parameters are visible to all instances and all processes across the Organization. Although business parameters may be changed every once in a while, they are not meant to be used as variables. Rather, they provide a way of storing long-lived values, such as a sales tax rate, without having to hard-code them into Process Business Language methods.

### Organizational Units

Organizational units are typically departments or divisions within an organization. Organizational units can be organized in a hierarchy.

For example:



In this hierarchy, Dallas is a single top-level organizational unit which contains the Customer Support, Documentation, and Product Management organizational units, while Customer Support contains Training, Technical Support, and Product Support organizational units.

Once the organizational units have been defined, participants may be assigned to one of the organizational units in the hierarchy. Processes can be deployed for one of the organizational units defined so that only participants in that organizational unit and in lower levels within the hierarchy are able to perform tasks in a process.

Every organizational unit might have a different calendar rule associated to it. This allows the Process Execution Engine to take into account time zones and working schedules set for the organizational unit where processes are deployed and to calculate deadlines accordingly.

Studio allows you to define the organizational hierarchy and the properties of each organizational unit. Remember that all the changes introduced to the organizational structure require a **Refresh Engine Data** operation if they are to be made available to processes on a currently running Process Engine.

## Roles

A role in the organization is a title or job function which is associated to a set of activities performed by participants of the organization.

Examples of roles include Accounts Manager, Sales Clerk, or Customer. Roles are similar to job titles, but are more flexible because a participant can be assigned to several roles, and some roles, such as Customer, may not be jobs at all.

## Roles and Activities

Every interactive activity is defined under a role. This is done by placing the activity within a *swim lane* with the name of the role. Swim lanes with no role name are only used for automatic activities which require no user interaction, and are not assigned to a role.

## Roles and Participants

Participants are assigned one or more roles. This is how the process can determine which participants can execute a given activity.

See [Permissions and Security in an Organization](#) for more information.

## Parametric Roles

A role can be defined as *parametric*. A parametric role includes a parameter which can adopt one of a set of values defined with the role.

For example, the role could be called "sales support" and the parameter could define a set of regions, such as East, West, and South. Even though there is only one role from a functional point of view, participants are assigned based on the location parameter.

Parametric roles require an instance because the parameter to be used is defined as an instance variable. This means that global activities cannot be assigned to parametric roles.



**Restriction:** Global activities cannot be assigned to parametric roles.

## Groups

Groups are collections of roles. In this way, it is possible assign multiple roles to participants in a single step. Groups may also contain other groups.

Unlike an organizational unit, which can belong to only one parent organizational unit, a group may be included in many other groups. Groups are therefore not organized in a hierarchical structure. However, if a group is included in another group, then it cannot have as a member that group. That is, so long as group A includes group B, group B cannot include group A.

## Participants

Participants defined in the organization are all the people enabled to track and perform tasks of business processes designed and developed with Studio.

A participant might belong to an organizational unit. If so, he can only perform tasks on processes deployed in that organizational unit or any organizational units that are below it.

You can assign a set of roles to a participant. A participant who logs in to Workspace can perform all the tasks defined for the roles assigned to him.

You can create, edit, and delete participants from the **Project Navigator**. Participants are usually created within Studio for process design and testing purposes. When the process is implemented into production, actual participants will normally be imported from an existing company directory or will be defined within Process Administrator.

## Holiday Rules

Holiday rules are collections of holidays that can be applied to calendar rules.

Multiple holiday rules can be created as needed for different [Calendar Rules](#) on page 135. Holiday rules affect the available work days for participants and the scheduling of activity deadlines.

## Calendar Rules

Calendar rules define the work hours, time zone, and holiday rule assignment for organizational units.

Multiple calendar rules can be created as needed for different organizational units (such as day shift, night shift, east coast, west coast, etc.). Calendar rules determine the available work days for participants and the scheduling of activity deadlines.

## Business Parameters

Business parameters are used to store long-lived information defined at the organization level.

Information suitable for storage as a business parameter includes company address and phone data, tax rates used in calculations within the process, or infrequently changed economic values such as the prime lending rate. Business parameters are visible from any process within a project and should generally be thought of as constants, though they can be changed.



**Tip:** Business parameters should be used for infrequently changed values which you do not want to include in the actual code. For example, company address data, the prime lending rate, or a sales tax rate are all good uses for business parameters.

It is strongly recommended *not* to use Business Parameters for values which will change very frequently (once a day or more). For those cases consider other options.

If you do need to change a Business Parameter from Process Business Language code, you can change it at runtime using the component Business Parameter in the Lib category. See the Studio. component documentation.

- If you change a Business parameter from a method you must be aware that the new value is not immediately available for all instances. Even more, if this value is changed from a Process Business Language method, the result may not always be the expected one and not available at the same time across the all participants.
- If the business parameter is used in a due transition expression of an activity, the business parameter value that applies is the one defined at the time the instance enters the activity. For example, let's say the business parameter "MAXTIME" is used in the due transition expression of the activity "Reply to Customer". When the instance "Request Customer 1" arrives, the due time is calculated using the value that the MAXTIME has at that moment. If another instance (in any process) changes the value of MAXTIME or you manually change it in the Process Administrator, the new value does not apply for the due time of the instance "Request Customer 1" for the activity "Reply to customer". It will apply for all instances that arrive to that activity *after* the business parameter was changed.



**Note:** If you change the Business Parameter at runtime, and you then stop and restart the Studio Process Engine, all business parameters are restored from the project definition. However, the *Enterprise* Process Engine *does* maintain Business Parameter values through a start/stop cycle, because in a production environment Business Parameter changes are assumed to be permanent.

## Creating and Managing Organizations in Studio


### Creating a New Organizational Unit

You can add an Organizational Unit from the **Project Navigator**.

To add an organizational unit:

1. Expand **Organization** (🏢) in the **Project Navigator**.

**Organization** will expand to show **Organizational Units, Roles, Groups, Participants, Holidays, Calendars, and Business Parameters**.

2. If possible, expand **Organizational Units** ().  
Any existing organizational units will be listed. If you cannot expand, there are no organizational units present.
3. Right-click on either **Organizational Units** or on an existing organizational unit, and select **New** from the context menu.  
The **Name** dialog box will appear.
4. Input the name of the organizational unit and click **OK** to add it to the node where you obtained the context menu.  
the new organizational unit is either created under the root organization node in the tree (if you right-clicked on **Organizational Units**) or under the organizational unit you right clicked on. In this way, hierarchical organizations can be defined. An editor for organizational unit data will open.
5. In the organizational unit editor, you can add a description in the **Description** text box, and you can select a calendar. These are optional fields and you can edit them later if you wish.





**Note:** The calendar rule set to an organizational unit *does not* affect the time zone used to display information to participants belonging to that organizational unit. The time zone taken into account to display dates in WorkSpace is the one set in the WorkSpace **Settings** dialog box. Settings, including time zones, can be unique for each user.

## Creating a Role

You can add a regular or parametric role to the organization from the **Project Navigator**.

To create a role:


1. In the **Project Navigator**, expand the project where you want to create a role.
2. Expand **Organization** (.
3. Right-click **Roles** () , then select **New** from the context menu.  
The **Name** dialog box is displayed.
4. Enter a name for the new role, then click **OK**.  
The new role is created and an editor opens for the role.
5. In the editor, you can enter a label for this role in the **Label** field. The default value for the field is the name of the role, so changing it is optional.
6. In the editor, you can also enter a description for this role in the **Description** text box. This is optional.
7. If you want the new role to be parametric, click the **Parametric** checkbox. In the **Values** pane, add values as required with the **Add** button. You can remove unwanted values by selecting them and clicking on the **Remove** button.  
A parametric role must have at least one defined value. Otherwise will let you save the role, but reports an error.
8. Save the role.

The new role has been created.

## Creating a Group

You can add a group from the **Project Navigator**.

To add a group:

1. Expand **Organization** in the **Project Navigator**.
  2. If possible, expand **Groups** (.
- Any existing groups will be listed. If you cannot expand, no groups exist.



3. Right-click on **Groups**, and select **New** from the context menu.  
The **Group** dialog box will appear.
4. Input the name of the group in the **Name** field and click **OK** to add it.  
The new group is created. An editor for the group will open.
5. In the group editor, you can add a description in the **Description** text box. This is optional.
6. Save the group.

The new group has been created.

### Creating a Participant

You can add a participant from the **Project Navigator**.

To add a participant:

1. Expand **Organization** in the **Project Navigator**.
2. If possible, expand **Participants** (8).  
Any existing participants will be listed. If you cannot expand, there are no participants.
3. Right-click on **Participants**, and select **New** from the context menu.  
The **Participant** dialog box will appear.
4. Input the name of the participant in the **Name** field and click **OK**.  
The new participant is created. An editor for the participant opens.
5. Optionally complete the **First Name**, **Last Name**, and **Display Name** fields.
6. If the participant belongs to an organizational unit, select it from the **Organizational Unit** drop-down list.
7. Optionally complete the **E-mail address** field.
8. If you will use this participant in simulations, enter values in the **Efficiency** and **Cost per hour** fields.
9. Optionally set the Locale and Time Zone drop-down lists to values appropriate for the participant.
10. Add the groups the participant belongs to by clicking **Add** in the **Groups** pane, and selecting the desired group(s) from the **Groups** dialog box.
11. Add the roles the participant carries out by clicking **Add** in the **Roles** pane, and selecting the desired role(s) from the **Roles** dialog box.
12. Save the participant. If you close the editor without saving, the participant will still exist, but will not have any of the settings entered in steps 5 through 11.

### Creating a Holiday Rule

You can add a holiday rule from the **Project Navigator**.

To add a holiday rule:

1. Expand **Organization** in the **Project Navigator**.
2. If possible, expand **Holiday Rules** (10).  
Any existing holiday rules will be listed. If you cannot expand, no holiday rules exist.
3. Right-click on **Holiday Rules**, and select **New** from the context menu.  
The **Holiday Rule** dialog box will appear.
4. Input the name of the holiday rule in the **Name** field and click **OK** to add it.  
The new holiday rule is created, but it contains no holidays. An editor for the holiday rule will open.
5. In the holiday rule editor, you can add holidays by clicking **Add** in the **Roles** pane, and adding the desired holiday(s) from the **Holiday Rule** dialog box.
6. When you are done adding holidays, save the holiday rule.

The new holiday rule has been created.




**Note:** If you close the editor without saving, the holiday rule will still exist, but will not have any of the holidays added in step 5.

### Creating a Calendar Rule

You can add a calendar rule from the **Project Navigator**.

To add a calendar rule:

1. Expand **Organization** in the **Project Navigator**.
2. If possible, expand **Calendar Rules** ().  
Any existing calendar rules will be listed. If you cannot expand, no calendar rules exist.
3. Right-click on **Calendar Rules**, and select **New** from the context menu.  
The **Calendar Rule** dialog box will appear.
4. Input the name of the calendar rule in the **Name** field and click **OK** to add it.  
The new calendar rule is created, with default calendar values. An editor for the calendar rule will open.
5. In the calendar rule editor, mark the checkbox for each day of the week which is a work day in this calendar rule.
6. For each workday, set the first work period of the day, between the Starting Time and the Finish Time on the left side. A second work period can be specified by setting the Starting Time and Finish time on the right side. You control whether the second work period is enabled by setting the checkbox adjoining it.
7. Save the calendar rule.

The new calendar rule has been created.




**Note:** If you close the editor without saving, the calendar rule will still exist, but with the default calendar values.

### Creating a Business Parameter

You can add a business parameter from the **Project Navigator**

To add a business parameter:

1. Expand **Organization** in the **Project Navigator**.
2. If possible, expand **Business Parameters** ().  
Any existing business parameters will be listed. If you cannot expand, no business parameters are defined.
3. Right-click on **Business Parameters**, and select **New** from the context menu.  
The **Business Parameter** dialog box will appear.
4. Input the name of the business parameter in the **Name** field and click **OK** to add it. Business parameter names must be all upper case and the first character cannot be a number. Underscores are allowed.  
The new business parameter is created. An editor for it will open.
5. In the business parameter editor, select the data type of the parameter in the **Type** drop-down list. Data type choices are *Bool*, *Int*, *Real*, *Time*, *Decimal*, and *String*.
6. Enter a value for the whole organization in the **Organization Value** field. You can override this value for individual organizational units in the next step.
7. You can add values by organizational unit by clicking **Add** in the **Organizational Units** pane, and adding the desired organizational unit from the **Organizational Units** dialog box. For each organizational unit, specify a value in the **Values** column of the Organizational Units table.
8. Save the business parameter.

The new business parameter has been defined.

### Importing an Organization

You can import an organization separately from a project. This way, you do not need to setup your organization with every new project.

To import an organization

1. In the Project Navigator, expand your project so you can see **Organization** (🏢).
2. Right-click on **Organization** and chose **Import Data**.  
The **Open** dialog box will be displayed.
3. Choose an organization data file, and click **Open**.

The organization data will be imported.

### Exporting an Organization

You can export an organization separately from a project. The organization data will then be available for other projects.

To export an organization

1. In the **Project Navigator**, expand your project so you can see **Organization** (🏢).
2. Right-click on **Organization** and chose **Export** (📁).  
The **Save As** dialog box will be displayed.
3. An organization data file name will appear by default, with an XDMML extension. You can choose a different name or path, but you should keep the file type the same. Click **Save**.

The organization data will be exported to the file you specified.

## Using Organizations with the Embedded Process Execution Engine

When the embedded Process Engine is started from Studio (select **Run ► Start Engine** from the menu), all the information about the organization is copied to an isolated environment where the engine executes processes.


Changes introduced while the Engine is running will not be updated to the runtime environment until the Process Engine is stopped and started, or until a **Refresh Engine Data** operation is performed. Depending on how runtime engine properties are set, the **Refresh Engine Data** operation might be performed automatically after introducing changes to the organization structure.

See Engine Properties for further information on how the runtime environment is updated with the latest changes. Some changes might require users currently logged in to WorkSpace first log out before having the changes available in their WorkSpace sessions. Refer to Refreshing the Embedded Execution Engine Data for further details on this topic.



## Attribute Reference

The following sections describe the attributes of each element within an organization.


### Organizational Unit Attributes

Attribute	Description
Name	Displays the name of the organizational unit.  <b>Note:</b> This name is defined when you create an organizational unit. It cannot be edited later.
Description	Contains a description of the organizational unit.
Calendar	Allows you to select a calendar rule for this organizational unit.


**Role Attributes**

Attribute	Description
Name	Displays the name of the role.  <b>Note:</b> This name is defined when you create a role. It cannot be edited later.
Label	Allows you to define a custom label for this role. This label is displayed in the swim lanes of the process editor view and can be changed after a role is created.
Description	Contains a description of the role.
Parametric	Shows whether the role is parametric or not. Defining a role as parametric allows you to split a role into subroles. The subroles are determined by values added to the role definition.  <b>Note:</b> You can only define a role as parametric when it is created.
Value	Allows you to define the parametric values for a role.

**Group Attributes**

Attribute	Description
Name	Displays the name of the group.  <b>Note:</b> This name is defined when you create a group. It cannot be edited later.
Description	Contains a description of the group.
Roles	Displays the roles associated with this group. You can associate new roles by clicking the <b>Add</b> button. If a role is parametric, you can choose the parameter associated with this role by selecting it from a drop-down menu within the table.
Groups	Displays the groups associated with this group. To make assigning permissions easier, a group can contain a collection of subgroups. You can associate new groups by clicking the <b>Add</b> button.


**Participant Attributes**

Attribute	Description
Name	Displays the name of the participant.  <b>Note:</b> This name is defined when you create an participant. It cannot be edited later.
First Name	Defines the first name of this participant


Attribute	Description
Last Name	Defines the last name of this participant
Display Name	Defines the display name for this participant. This is the name that appears in Oracle BPM WorkSpace.
Organizational Unit	Allows you to specify which organizational unit the participant belongs to.
E-mail address	Defines the email address of the participant
Locale	Defines the locale for this participant. The locale value determines the language used in Oracle BPM WorkSpace.
Time Zone	Defines the time zone for this participant. The time zone value determines the time zone used in Oracle BPM WorkSpace.
Roles	Displays the roles associated with this participant. You can associate new roles by clicking the <b>Add</b> button. If a role is parametric, you can choose the parameter associated with this role by selecting it from a drop-down menu within the table.
Groups	Displays the groups associated with this participant. You can associate new groups by clicking the <b>Add</b> button.

### Holiday Rule Attributes

Holiday Rules allow you to define a collection of holidays and other non-work days. Holiday rules can be assigned to an organizational unit.


Attribute	Description
Name	Displays the name of the holiday rule.  <b>Note:</b> This name is defined when you create an holiday rule. It cannot be edited later.
Description	Displays a description of an individual holiday.
Type	Defines the type of holiday. Valid values are: <ul style="list-style-type: none"> <li>• Common</li> <li>• Fixed</li> </ul>
Date	Defines the date of the holiday.

### Calendar Rule Attributes

Attribute	Description
Name	Displays the name of the calendar rule.  <b>Note:</b> This name is defined when you create an holiday rule. It cannot be edited later.

Attribute	Description
Time Zone	Specifies the time zone for this calendar rule. This information is used by Oracle BPM WorkSpace to determine when a participant is available to perform work.
Holiday Rule	Specifies the holiday rule associated with this calendar rule.
Work Days	Defines the number of work days per week as well as the number of hours each day.

### Business Parameter Attributes

Attribute	Description
Name	Displays the name of the business parameter.  <b>Note:</b> This name is defined when you create the business parameter. It cannot be edited later.
Type	Defines the type of data the business parameter contains.
Organizational Value	Defines the value of the business parameter.
Organizational Units	Specifies the organizational units that use this business parameter. This table also allows you to define the value of the business parameter for each organizational unit.

## Simulations

### Simulation Overview

After creating a process model, Oracle BPM Studio allows you to run simulations to determine the performance of your process model. You can also use process simulations to compare how changes to an existing process will affect performance. You can run process simulations based on simulated data or real-world data from production processes.

Oracle BPM allows you to simulate the behavior of process models based on real or simulated data. After you have designed a business process model, Oracle BPM Studio allows you to run process simulations to determine their efficiency. You can also use simulations to test the effects of changes on your process design.

Process simulations do not execute each individual task within a process. For example, the code within an activities task is not executed, variables are not assigned values, and external resources are not updated. However, you can mimic the behavior of these elements of process's activity by configuring different attributes within a simulation model. These attributes include:

- duration
- resources
- costs
- transitions

## Process Simulation Models

To execute a process simulation, you must define simulation models. Simulation models allow you to specify behavior of each element of your process. There are two types of simulation models in Oracle BPM Studio:

Simulation Model	Description
Process Simulation Models	Allow you to define the behavior for an individual process. See <a href="#">Process Simulation Model</a> on page 143.
Project Simulation Models	Allow you to define the behavior for an entire project. A project simulation model is composed of a group of process simulation models. Within a project simulation models you can choose which process simulation models to run. See <a href="#">Project Simulation Models</a> on page 144.

You can configure multiple simulations models for a project and its processes. This allows you to mimic different combinations of resources, etc.

## Project Simulation Models

A project simulation model functions as a container for process simulation models. A project simulation model allows you to define a scenario for an entire project. It also allows you to determine how processes work together

## Simulations View


After defining your process and project simulation models, you can run and view simulations using the Simulations View. The Simulations View provides controls for starting, stopping, and pausing simulations. It also provides controls for displaying simulation data

## Simulations Editor

Once you start a simulation, the Process Editor window displays the path of the simulated in-flight instances.

## Process Simulation Model

Process simulation models allow you to define how a process behaves as part of a Project Simulation Model. You can define multiple process simulation models for each process. This allows you to create different simulations based on different combinations of resource allocations and activity behavior.

 **Note:** Grab, Global, and End activities are not simulated.

## Process Information Tab

The Process Information Tab allows you to configure in:

Option	Description
Enable amount of current instances	Allows you to define the number of instances that can exist within the simulation at one time. The process simulation will run until the duration is completed or the maximum number of instances is reached.

The following table lists the Distribution Types used when creating process instances:

Option	Description
Constant	Generates simulated process instances regularly as defined by the period property.
Uniform	Generates simulated process instances regularly, taking into account the variation specified in the delta property.

Option	Description
Exponential	Creates simulated process instances using an average frequency of instances within a specific interval.
Normal	Generates simulated processes according to a Gauss Bell distribution based on a mean and standard deviation.
Real	Creates simulated process instances based on specific time-based criteria. This type of distribution is primarily used with round-trip simulations. You can specify the interval criteria used to categorize the distribution. You can also specify the mean and standard deviation.

## Project Simulation Models

A Project simulation model defines the behavior of the simulation for the entire project.

Project simulation models allow you to customize the following parameters to see how they influence the performance of your project:

- Start time and duration of the simulation
- Determine which process simulation models you want to include in the project simulation
- Allows you to define the participant resources you want to include in the simulation
- Allows you to define the priority distribution of instances within the simulation

Within a project, you can define multiple project simulations. Defining different project simulations models allows you to test different combinations of resources and priorities. The following parameters can be configured for a project simulation model.

Parameter	Description
Start Time	Defines the start time for the simulation. This time is used only for logging. It is not used for scheduling purposes.
Duration	Defines the period the simulation will run. This interval is specified in months, days, hours, minutes, and seconds.
Use Calendar Rule	Determines if calendar rules are used in simulation. Checking this box allows the simulation to account for calendar rules when determining participant allocations.

Project Simulation Models also contain the following tabs which allow you to further define your simulation.




### Project Tab

The Project Tab contains a table that lists all of the processes within the current project. For each process, you can select which process simulation model you want to use for each process. Also, you must specify which processes to include in the simulation.

For all of the included processes, instances are generated when the simulation is run.

### Resource Tab

The Resource Tab allows you to define the resources used within the process simulation. All process included in the simulation will share these resources. The cost of each resource is defined per hour.

Icon	Description
	Loads resources from the organization into the project simulation model.
	Adds a new resource to the project simulation model.
	Deletes the currently selected resource from the project simulation model.



## Priority Tab

The Priority Tab allows you to specify the probability for priority distribution of an instance. This priority determines the way an instance flows within a process. The sum of all priority distributions must equal 100.

## Creating and Running a Process Simulation Model

The following tasks show you how to create simulations models and run a simulation.

### Creating a Process Simulation Model

The following steps describe how to create a Process Simulation Model.

1. In the Project Navigator View, expand the Project where you want to create the Process Simulation Model.
2. Expand **Processes**.
3. Right-click on the Process.
4. Select **New Process Simulation Model**.
5. Enter a name for you new Process Simulation Model.
6. Click **OK**.

The Process Simulation Model appears in the editor window. It also appears as a Resource in the Project Navigator View.

You can define the behavior of your Process Simulation Model.

### Creating a Project Simulation Model

The following steps describe how to create a Project Simulation Model.

You should define any Process Simulation Models that you want to include as part of your new Project Simulation Model. See [Creating a Process Simulation Model](#) on page 145.

1. In the Project Navigator View, right-click **Simulations**.
2. Select **New Simulation**.
3. Enter a name for your Project Simulation Model.
4. Click **OK**.
5. The Project Simulation Model appears in the editor window. Each Process Simulation Model that you have defined appears in the table.

The Project Simulation Model also appears as a Resource under Simulations in the Project Navigator View.

6. Using the drop-down menu, select **Yes** in the **Include in Simulation** for each Process Simulation Model that you want to include in your Project Simulation.

### Running a Simulation

The steps in this task outline how to run a Process Simulation based on Process and Project Simulation Models.

Ensure that you have created Process Simulations Models and at least one Project Simulation Model. See [Creating a Process Simulation Model](#) on page 145 and [Creating a Project Simulation Model](#) on page 145.

1. In the Project Navigator View, expand **Simulations**.
2. Select the Project Simulation Model that you want to run.
3. Ensure that the Simulation View is visible.

See [Simulation View Reference](#) on page 147.



**Note:** It may take several seconds for the Simulation View to load.

4. Click the Start icon.

The simulation animation plays in the Process Editor window. Simulation data is updated in the Simulation View.

## Round-trip Simulations

### Round-trip Simulations

Round-trip Simulation provides a way to create process simulation models based on real-world data. When creating a process simulation model using round-trip simulation, data is imported from the Process Execution Engine Database. You can use data imported from the following environments:

Process Execution Engine Database	Uses
Studio	Using data from Studio's process execution engine, you can quickly create a process simulation model.
Enterprise	Using data from the Enterprise process execution engine allows you to create process simulation model based on 'real-world' data. You can use this data to create a base-line simulation that mimics the performance of your processes. This base-line can be compared to simulations of revised processes to determine how performance can be improved.



**Note:** After creating a process simulation model using round-trip simulation, you must manually enable it in your Project Simulation Model. See [Creating a Project Simulation Model](#) on page 145.

### Running a Round-trip Simulation in Studio

This topic outlines the procedures for running a round-trip simulation using within Studio. For testing purposes, you can create a round-trip simulation model using WorkSpace to generate data based on real world situation. This data is then imported from the Embedded Process Execution Engine into Studio.

Before running the following procedures, ensure that you have created a project and at least one process that you want to simulate.

1. Start the Embedded Process Execution Engine within Studio.
2. Run WorkSpace
3. Use your deployed process to generate simulation data.

You should create and use enough instances of your process to create meaningful test data. There is no minimum threshold of data. The exact amount of data required depends on the process you are simulating.



**Note:** Before continuing to the next step, you must wait several minutes for the embedded process engine database to update.

4. Right-click on your Project.
5. Select **Extract Simulation**.
  - a) Provide a name for the new Process Simulation Model
  - b) Select the Process you where you are creating the simulation model.
  - c) Select the distribution criteria for this simulation.
6. Click **OK**.


## Simulation Reference

### Simulation View Reference

The simulation view allows you to run and view simulations.

After defining Project and Process simulation models, you can run a Project simulation from the Simulation View.

The Simulation View toolbar allows you to perform the following actions:

Toolbar Element	Description
Play Icon	Starts the simulation. If the processes included in your simulation are not open in an editor window, Studio opens them.
Stop Icon	Stops the simulation.  <b>Note:</b> If you stop a simulation, you must restart it from the beginning.
Pause Icon	Pauses the simulation. You can press the start icon to resume the simulation.
Run to End Icon	Runs the simulation in the background with no animation. This allows you to run the simulation faster.
Simulation Speed	Determines how fast simulated process instances are created. In normal speed, instances are created at rate of one per second.

### Log Tab

The Log Tab displays a log of all the actions performed during the simulation.

### Process Simulation Model Reference

Each process simulation model allows you to define parameters for the simulation as a whole as well as define parameters to mimic the behavior of the activities within the process.

### Activities Tab

Each configurable activity contains tabs that allow you to edit different simulation properties. The following table shows what process parameters you can edit for each activity type.

	Duration	Cost	Queue Info	Resources	Transitions	Inner Activities	Copies	Related Process
Automatic	x	x	x		x			
Begin					x			
Conditional					x			
Decision	x	x	x	x	x			
Group	x	x	x	x	x	x		
Interactive	x	x	x	x	x			
Join					x			

	Duration	Cost	Queue Info	Resources	Transitions	Inner Activities	Copies	Related Process
Multiple					x		x	
OrSplit					x			
Process Creation	x	x	x		x			
Subflow	x	x	x		x			x
Termination Wait	x	x	x		x			

### Duration

The **Duration** tab allows you define the amount of time required to complete the simulated activity.

### Constant Distribution

Constant distribution causes the simulated time to complete an activity to be determined based on the **Period** property described below.

Parameter	Description
Period	Determines the period required to complete an activity.

### Uniform Distribution

Uniform distribution determines the period required to complete an activity consistently, taking into account the variation specified in the **delta** property.

Parameter	Description
Mean	Determines the mean time it takes to complete an activity.
Delta	Defines the upper and lower limit variation of the <b>mean</b> parameter when determining how long it takes to complete a simulated activity.

### Exponential Distribution

Exponential distribution determines how long it takes to complete a simulated activity by specifying how many instances are completed within a specific period.

Parameter	Description
Average Frequency	Determines the number of average instances processed within the interval defined by the <b>Every</b> property.
Every	Defines the interval used for exponential distribution.

### Normal Distribution

Normal distribution uses the Gauss Bell distribution to determine how long a simulated activity takes to complete. You must specify the mean and standard distribution.

Parameter	Description
Mean	The mean period required to perform an activity.
Standard Deviation	The standard deviation of the mean required to perform an activity.

### Real Distribution

Real distribution allows you to specify the amount of time required to complete a simulated activity for a specific time interval.

Parameter	Description
Distribution Criteria	Determines the time interval for determining how long a simulated activity takes to complete.
Interval	
Mean	Defines the mean time to complete an activity.
Standard Deviation	Defines the standard deviation of the mean parameter.

### Cost

The **Cost** tab allows you define the amount of resources required to complete the simulated activity.

Parameter	Description
Fixed Base Cost	Defines the cost required to perform the simulated activity.
Fixed Base Cost Plus Resource Cost	Calculated based on the define cost per hour and the time it takes the resource to execute the instance.

### Queue Info

The **Queue Info** tab allows you to configure the simulated behavior of how process instances are queued for a given activity.

Parameter	Description
Queue Warning Size	Determines the number of incoming instances that can be waiting for an activity at a time.
Activity Queue Policy	Determines how incoming instances are handled by the activity. The following values are available: <ul style="list-style-type: none"> <li>• F.I.F.O.</li> <li>• L.I.F.O.</li> <li>• Random</li> <li>• By Priority</li> </ul>

### Resources

Parameter	Description
Use Organization Resource	Uses resources defined as part of the organization of the project.
Participant Selection Policy	Can be based on the Minimum Cost, Maximum efficiency or Randomly. Cost and Efficiency values

Parameter	Description
	are those defined in the project simulation model definition for each participant.
Use Fixed Resources	Indicates the number of participants assigned to the Interactive activity. This option is used when costs and efficiency parameters are not relevant in the evaluation but only the amount of resources is needed.
Available Resources	Specifies a fixed number of resources available.

**Transitions**

The **Transition** tab determines how an activities transitions are handled in the simulation.

Parameter	Description
Transition	Lists all of the outgoing transitions of an activity.
Probability	Determines the probability that a transitions will be executed.

**Inner Activities**

The **Inner Activities** tab allows you to select whether the process simulation model will simulate activities within an activity group.

**Related Processes**

The **Related Processes** tab allows you to configure whether the activities within a subflow are included within the process simulation.

Components Catalog

About Components

Components in Oracle BPM are object types containing data (attributes) and behavior (methods).



Components can be defined in Oracle BPM's PBL language (BPM Objects) or cataloged from external technologies (such as Java, SQL tables or Web Services). Oracle BPM also provides a standard library of components for common BPM programming tasks.





**BPM Objects**

You can create new components in your project using the PBL language. These components are called BPM Objects and may be composed of other BPM Objects and external components defined in other technologies.

Use BPM Objects to define high-level business concepts and logic which can be re-used across your project. BPM Objects provide a common programming layer to integrate lower-level components provided by heterogeneous technologies (such as Java, Web Services and SQL databases).

You can create BPM Objects of the following types:

Component	Description
 General BPM Object	A user-defined component that contains <i>attributes</i> , <i>methods</i> , and <i>presentations</i> .
 BAM Dashboard	A special type of BPM Object for Business Activity Monitor presentation

Component	Description
 Enumeration	A special type of BPM Object that represents a data type with a fixed set of possible values.
 Business Exception	A special type of BPM Object that represents an exception triggered by a foreseeable business condition.
 PUnit Suite	A special type of BPM Object that represents a unit test suite for processes.
 CUnit Suite	A special type of BPM Object that represents a unit test suite for individual BPM Objects.

### External (cataloged) Components

With Oracle BPM you can leverage external APIs (Application Programming Interfaces), services and data sources. To use external components in your processes you must first include them into your project catalog. You can catalog components exposed in different technologies such Java, .Net, COM, SOAP Web Services and others.

Once a component has been added to the catalog, it can be integrated into the project processes, thus enabling you to utilize existing applications or services. For example, you can catalog components of an existing SQL database and then use them to query or modify information contained in the database.

### Standard Components

Oracle BPM includes a set of standard components to facilitate common BPM programming tasks. Components under the **Java** module expose the standard Java APIs. Components under the **Fuego** module provide BPM-specific components (Refer to the *Oracle BPM Components Reference* for details).

The standard components are always available and are predefined in the component catalog of every BPM project.

## About the Components Catalog

The components catalog of a BPM project defines the set of business objects and services available to all business processes in the project.

The catalog organizes components in a hierarchical structure of *modules*. Each module can contain components and other modules.

### Standard Modules

The following standard modules are included in every project:

- **Fuego**: Contains sub-modules with BPM-specific components. Refer to the *Oracle BPM Components Reference* for details.
- **Java**: Contains the complete standard Java APIs. You can use the Java APIs directly from PBL.
- **Plumtree**: Contains legacy APIs for integration with Oracle User Interaction (formerly AquaLogic Interaction).

The standard modules are always available and are predefined in the component catalog of every BPM project.

The standard modules are read-only. You cannot remove, modify or add components (or sub-modules) to the standard modules.

## User-Defined Modules

You can add your own modules to the project catalog. You can also add modules within modules, organizing them in a hierarchical structure.

You can create new BPM Objects and catalog external components into your modules. External components provide a way to communicate with external applications, services, APIs, databases, or other software resources your project needs to exchange information with.

## Creating a Module

Ensure that the Project Navigator View is visible and that you have created or opened a Project.

1. Right-click and select **Catalog** (🔍) ► **New** ► **Module** .

To create a new module inside an existing one, right-click on the existing module (🔍) instead.


2. Enter a name for the new Module, then click **OK**.  
The new Module appears under the **Catalog** resource.

You may Right-click the new Module to get the list of available operations on that Module.

## Deleting a Module

Ensure that the Project Navigator View is visible and that you have created or opened a Project.

1. Right-click on the module you want to delete and select **Delete** (✖) .

 **Important:** Deleting a module also removes all elements inside that module. This operation cannot be undone.

2. Confirm whether you want to proceed.

 **Note:** This confirmation dialog may be disabled.

## External Components

External components are those you catalog from external technologies, such as Java, SQL and Web Services.

With BPM Studio you can leverage external APIs (Application Programming Interfaces), services and data sources. To use external components in your processes you must first include them into your project catalog.

Cataloging services from different technologies into BPM components provides your project with a common programming model, freeing developers from the integration and conversion details between otherwise incompatible technologies. For example, you can read data from a Web Service, process it using an external .Net component and store the results on a database using a SQL component.

### External Resources Configuration

To catalog external components you need to provide the necessary configuration information to locate them. For example, to catalog Oracle database tables you must supply information such as the host, port, user and password to access the server.

This configuration information becomes part of the project under [External Resources](#) on page 198 in the Project Navigator. Each external component in your catalog is associated with an External Resource.

Studio uses External Resources at cataloging time to inspect the components' definitions, and on runtime to initiate connectivity and execute component invocations.



## .NET Components

You can catalog .NET components and use them in your BPM project. Microsoft® .NET is a multi-language software development and execution framework for Microsoft Windows.

### Integrating .NET Assemblies

Assemblies are the building blocks of .NET Framework applications. An assembly is a collection of types and resources that are built to work together and form a logical unit of functionality. An assembly provides the common language runtime with the information it needs to be aware of type implementations. To the runtime, a type does not exist outside the context of an assembly.

If want to call an external application that exposes itself using .NET Assemblies, you need access to the .NET documentation provided with that application (or contact the software vendor for further information).

### Cataloging a .NET Component

Before using .NET components from BPM processes and objects, you must include them into the project catalog. When you catalog a .NET assembly, you are gathering all the necessary information that Studio needs in order to call and execute it at runtime.

Before cataloging .NET components, ensure the [About the .NET Bridge](#) on page 153 is running on the machine hosting the .NET assemblies.

Also ensure that you have created a module where you want to catalog the .NET components. See [Creating a Module](#) on page 152.

To catalog .NET components:

1. Right-click on the module where you want to catalog the components.
2. Select **Catalog Component ► .NET Component**
3. Select one of the following options:

Option	Description
<b>Use existing configuration</b>	Select this option if you have already configured an External Resource for Microsoft .NET Service.
<b>Create a new configuration</b>	Select this option if you need to configure a new External Resource for Microsoft .NET Service. See <a href="#">Creating an External Resource</a> on page 199.

4. Click **Next**.  
Oracle BPM Studio connects to the .NET Bridge and presents you with a file system browser of the host running the .NET Bridge.
5. Select the .NET assembly file containing the components you want to catalog.  
Browse and select the .dll file that corresponds to the .NET assembly you want to inspect, and click **Next**. Not every .dll file is .NET file. If you have selected an invalid .dll file, an error is displayed. Click **Back** to select the .NET .dll file again.
6. Click **Finish** to catalog the .NET components.  
The components contained in the selected .NET assembly are now included in your catalog.

### About the .NET Bridge

Oracle BPM provides the .NET Bridge application that allows you to catalog and execute .NET components from BPM projects.

The .NET Bridge (`netbridge.dll`) is a Windows application that acts as a bridge between BPM applications and .NET Assemblies. Oracle BPM supplies this application to provide all the necessary services to catalog and use .NET components.

The .NET Bridge runs as a standalone process on the Windows server hosting the assemblies. The bridge is itself a .NET application.

On runtime, Oracle BPM connects to the .NET Bridge through a TCP port (by default 5050). Therefore, the Oracle BPM Process Execution Engine can run in a different host environment from the one where .NET Bridge is running (e.g., your BPM Engine may reside on a Solaris box and leverage .NET components running on a separate Windows server).

All components called by the bridge share the same CLI (Common Language Infrastructure) as the bridge itself since they are called using `System.Reflection` APIs (it follows that they share the same process).

**COM Components**

You can catalog COM components and use them in your BPM project. COM components are software programs that use the Microsoft Component Object Model (COM).

COM objects can be created in several different ways and with different tools, such as Visual Basic or C++. Several applications expose functionality as COM objects - this includes most of Microsoft's applications (Office, Internet Explorer, etc.) and many third-party applications.

**Integrating COM Applications**

If you want to call an external application that exposes itself using COM, you need access to the COM documentation provided with that application. For example, if you wish to call Microsoft Excel components, you will need to understand Excel's object model, which is documented in the Microsoft Developer Network Website.

**Cataloging COM Components**

Before using COM components from processes or BPM Objects, you must include them into the project catalog. When you catalog a component, you are gathering all the necessary information that Studio needs in order to call and execute it at runtime.

Before performing the procedures in this task, you should ensure that the [About the COM Bridge](#) on page 158 is installed and running on your system. See [Installing COM Bridge as a Service](#) on page 158 for more information.

You should also ensure that you have created a module where you want to catalog the COM components. See [Creating a Module](#) on page 152.

To catalog a COM Component:

1. Right-click on the module where you want to catalog the COM component.
2. Select **Catalog Component ► COM**
3. Select one of the following options:

Option	Description
<b>Use existing configuration</b>	Select this option if you have already configured an External Resource for the COM Component.
<b>Create a new configuration</b>	Select this option if you need to configure a new External Resource for the COM Component. See <a href="#">Creating an External Resource</a> on page 199.
4. Click **Next**.  
Oracle BPM Studio creates a list of COM Type Libraries that can be cataloged. This may take several minutes.
5. Select the COM Type Libraries you want to introspect.
6. Click **Next**.  
The libraries are analyzed for dependencies and introspected.
7. Click **Finish**.

The libraries you chose to catalog appear in the Project Navigator.

## COM example with MS Word

To run this example you have to catalog the Microsoft Word Object Library.

### Creating a New Word File

The PBL below, shows how to create and fill with text a Word file.

```
filename = "C:\\WordTest" + id.number + ".doc"

Application.visible = false

// get ready to use Word
worddocs = Application.documents

// create a new Word document
worddoc = add(worddocs)

// find something specific in the document.
// in this case there isn't anything,
// but the rangevar variable is still needed
// for the next command which inserts the text.
rangevar = range(worddoc)

// tell Word where you want to put the sentence.
// I say after rangevar, but as you know from above,
// that isn't very specific.
insertAfter rangevar
    using text = sentence + "\nFuego is the
        greatest software ever!"

// save as the filename you want.
// We built the name somewhat unique in a variable
saveAs worddoc
    using fileName = filename

// close the newly saved document
close worddoc

// quit
quit Application
```

### Setting Values in an Existing Word File

The PBL below, shows how to complete form fields in a Word file used as template, and saved as a new file.

```
// Initialize variables
custName = customerName
worddocs = wordappl.documents
Application.visible = false

// Open Word file
open worddocs using
    fileName = "C:\\tmp\\input.doc" returning worddoc

// Initialize the form fields into their object
wordformfields = worddoc.formFields

// Select the form field you want to work on
item wordformfields using
    index = "Text2" returning wordformfield

// Set the value of the form field you selected
wordformfield.result = custName
```

```
// Save as a different Word file
saveAs worddoc using
    fileName = "C:\\tmp\\result.doc"

// Quit Word
quit wordappl
```

### COM example with MS Excel

To run this example you have to catalog the Microsoft Excel Object Library under a module named MSEXCEL.

The PBL code shows how to use Excel from the BPM system.

```
do
// Open the sheet open MSEXCEL.Application.workbooks using
"C:\\tmp\\invoice.xls"

// Set it visible (for debugging purposes)
// MyExcel.Application.visible = true

// Get the active sheet cells
cells = Worksheet(
    MSEXCEL.Application.activeWorkbook.activeSheet).cells

// Ask for some client data, this could be taken
// from a database
input "Name:" name,
    "Address:" address,
    "State: " state,
    "Zip: " zip,
    "Phone:" phone,
    "Order No.:" orderNo
    using title = "Enter product"

// Fill the sheet, the getItem method, takes two args
// (in the case row and column), and returns a
// MSEXCEL.Range object that matches the criteria
// for selection
// Fill Name

getItem cells using 4, 3 returning temp
cell = MSEXCEL.Range(temp)
cell.value = name

// Fill Date
getItem cells using 4, 9 returning temp
cell = MSEXCEL.Range(temp)
cell.value = 'now'

// Fill Address
getItem cells using 5, 3 returning temp
cell = MSEXCEL.Range(temp)
cell.value = address

// Fill State
getItem cells using 5, 5 returning temp
cell = MSEXCEL.Range(temp)
cell.value = state

// Fill Zip
getItem cells using 5, 7 returning temp
cell = MSEXCEL.Range(temp)
cell.value = zip
```

```

// Fill Order No.
getItem cells using 5, 9 returning temp
cell = MSEXCEL.Range(temp)
cell.value = orderNo

// Fill Phone
getItem cells using 6, 3 returning temp
cell = MSEXCEL.Range(temp)
cell.value = phone

// Now we enter some data about products
for i in 10..21 do
    qty = 0
    product = ""
    price = 0

    // We ask for it

    input "Quantity:" qty, "Product:" product, "Unit Price:" price
        using title = "Enter product", buttons = ["Ok", "Finish" ]
    returning buttonPressed = selection

    exit when buttonPressed != "Ok"

    // And then we fill the sheet
    // Fill Qty.

    getItem cells using i, 2 returning temp
    cell = MSEXCEL.Range(temp)
    cell.value = qty

    // Fill product
    getItem cells using i, 3 returning temp
    cell = MSEXCEL.Range(temp)
    cell.value = product

    // Fill price
    getItem cells using i, 8 returning temp
    cell = MSEXCEL.Range(temp)
    cell.value = price
end

// We finally ask for the shipping cost

price = 0
input "Shipping cost" price
    using title = "Invoice"

// Fill Shipping
getItem cells using 23, 9 returning temp
cell = MSEXCEL.Range(temp)
cell.value = price

display "What do you want to do with this sheet?"
    using title = "Invoice finished",
        options = ["Preview", "Print"],
        default = "Preview"
    returning buttonPressed = selection

if buttonPressed == "Print" then
    //Print it
    printOut Worksheet(
        MSEXCEL.Application.activeWorkbook.activeSheet)

```

```
else
    //Preview it
    MSEXCEL.Application.visible = true
    printPreview Worksheet(
        MSEXCEL.Application.activeWorkbook.activeSheet)
end

//Mark it as saved
MSEXCEL.Application.activeWorkbook.saved = true

// Just in case, we ask Excel to quit
quit MSEXCEL.Application

end
```

**About the COM Bridge**

Oracle BPM COM Bridge allows you to catalog and use COM components in Oracle BPM processes. BPM COM Bridge is a Windows application that acts as a bridge between Oracle BPM applications and COM. Oracle BPM supplies this application to provide all the necessary services to catalog and use COM components. BPM COM Bridge is packaged as a separate application and runs as a standalone Windows process. The Bridge should be installed on the machine where the COM components reside. However, this is not absolutely necessary for all components because DCOM (Distributed COM) components, if properly configured, can be located in a different machine from the one that is running COM Bridge. On runtime, Oracle BPM connects to the COM Bridge through a TCP port (by default 4042). Therefore, the Oracle BPM Process Execution Engine can run in a different host environment from the one where COM Bridge is running (e.g., your BPM Engine may reside on a Solaris box and leverage COM components running on a separate Windows server).

**The COM Bridge executable**

There are two versions of Oracle BPM COM Bridge:

combridge.exe	This program is installed with Studio and runs automatically when required. It is intended for the development stage of a project.
combsvc.exe	Is the Windows service version of BPM COM Bridge. It is intended for production environments, where no GUI interaction is required with the COM components or applications that are being automated. After installation, it will start whenever the machine starts.

**Installing COM Bridge as a Service**

You can install Oracle BPM COM bridge as a Windows Service, so that it automatically starts when Windows boots.

The COM Bridge runs as an operating system service. It is only supported on Windows environments. The COM Bridge application is included with Oracle BPM Studio and Enterprise.

1. To install COM Bridge as a Windows service simply run <ORABPM\_HOME>/bin/combsvc -install.  
A new Service is created on your Windows system.
2. Start the service.  
You may start the COM Bridge service from the standard Windows Services panel. Alternatively, you may start it from the command line using: <ORABPM\_HOME>/bin/combsvc -start

**COM Bridge options**

The COM Bridge application and service provide different command-line options for installing, starting and stopping the bridge.

**Standalone Version**

Command	Description
combridge -install	Installs the COM Bridge to start on every login.
combridge -remove	Uninstalls the COM Bridge.
combridge -debug	Disables asynchronous logging.
combridge -stop	Stops another COM Bridge running.
combridge -log filename	Sets the log file name (default is %TEMP%\COMBridge.log).
combridge - ?	Displays COM Bridge usage dialog
combridge [option] [port]	The port is the TCP port where COM Bridge is listening to incoming calls. Defaults to port 4042.

**Service Version (recommended for production)**

Command	Description
combsvc -install	Installs the COM Bridge as a service.
combsvc -remove	Uninstalls the COM Bridge as a service.
combsvc -debug [params]	Run the COM Bridge as a console application for debugging.
combsvc -start	Starts the COM Bridge service.
combsvc -stop	Stops the COM Bridge service.
combsvc - ?	Displays COM Bridge service commands.

**CORBA Components**

You can catalog CORBA components and use them in your BPM project. Studio allows you to catalog CORBA objects that reside in a CORBA Interface Repository.

**Supported CORBA environments**

JCorb (not included) is the client library supported by Oracle BPM to integrate with CORBA components.

Oracle BPM supports integration with JCorb and Orbix CORBA Servers.

There is no support for CORBA name servers.

**Supported CORBA Objects**

Any CORBA type can be cataloged through Studio into the project catalog. However, the following are the only types supported at run time:

- Interfaces with attributes and operations
- Structs
- Unions
- Sequences
- Enumerations
- Arrays
- Aliases

## Cataloging a CORBA Component

Before using CORBA components from processes or BPM Objects, you must include them into the project catalog. When you catalog a component, you are gathering all the necessary information that Studio needs in order to call and execute it at runtime.

Before performing this procedure, ensure you have created a module where you want to catalog the COM component. See [Creating a Module](#) on page 152.

1. Right-click on the module where you want to catalog the CORBA components.
2. Select **Catalog Component ► CORBA Service**
3. Select one of the following options:

Option	Description
<b>Use existing configuration</b>	Select this option if you have already configured an External Resource for the COM Component.
<b>Create a new configuration</b>	Select this option if you need to configure a new External Resource for the COM Component. See <a href="#">Creating an External Resource</a> on page 199.

4. Set the Naming Service Values.
5. Set the Interface Repository IOR.
6. Select the source of the IDL Definitions.
7. Select the Components.
8. Click **Next**.

## CORBA Array Examples

The following examples illustrate different ways to use arrays with CORBA objects.

### Unidimensional Arrays

Unidimensional arrays are supported with CORBA in BP-Methods. An example of arrays is shown in the following IDL.

```
interface ArrayTest
{

typedef string  StringArray[];
typedef long    LongArray[];
typedef boolean BooleanArray[];

struct TestStruct
{

long    longMember;
string  stringMember;

};

union TestUnion switch(boolean)
{

case TRUE: long longMember;
case FALSE: string stringMember;

};

typedef TestStruct TestStructArray[];
```



```
typedef TestUnion  TestUnionArray[];

};
```

### arrays as IN arguments

```
arrayTest = Module1.CorbaTests.ArrayTest("ArrayTest")

stringArray = [ "one", "two", "three",
                "four", "five" ]

stringArrayInOp arrayTest
    using aStringArray = stringArray
    returning stringArray

display stringArray
```

### Arrays as OUT Arguments

```
arrayTest = Module1.CorbaTests.ArrayTest("ArrayTest")

stringArray = [ "one", "two", "three",
                "four", "five"]

stringArrayOutOp arrayTest
    returning stringArray = aStringArray

display stringArray
```

### Arrays as INOUT Arguments

```
arrayTest = Module1.CorbaTests.ArrayTest("ArrayTest")

stringArray = [ "one", "two", "three",
                "four", "five"]

stringArrayInoutOp arrayTest
    using aStringArray = stringArray
    returning stringArray = aStringArray
```

### CORBA Enumeration Examples

CORBA enumerations can be used in many situations. The following examples show how to use the enumeration that appears in the Project Catalog after the IDL is cataloged.

#### Using enums in Operation Invocations

The following example shows how enumerations can be used in operation invocations or as the discriminator of a union:

```
module CorbaTests
{
    interface EnumTest
    {
        enum Slot { s1, s2, s3 };
    }
}
```

```

union UnionTest switch(Slot)
{

case s1: string    stringMember;
case s2: long      longMember;
default: boolean   booleanMember;

};

void enumOp(in Slot aSlot);

void unionOp(in UnionTest aUnionTest);

Slot retEnumOp(in Slot aSlot);

void outEnumOp(out Slot aSlot);

void inoutEnumOp(inout Slot aSlot);
};
};

```

### Using enums as IN Arguments

```

s1 = Module1.CorbaTests.EnumTest.Slot.s1

enumTest = Module1.CorbaTests.EnumTest("EnumTest")

enumOp enumTest using aSlot = s1

```

### Using enums as OUT Arguments

```

enumTest = Module1.CorbaTests.EnumTest("EnumTest")

outEnumOp enumTest returning s3 = aSlot

```

### Using enums as IN/OUT Arguments

```

enumTest = Module1.CorbaTests.EnumTest("EnumTest")

inoutEnumOp enumTest using aSlot = s2 returning s4 = aSlot

```

### enum as the Return Type

```

s2 = Module1.CorbaTests.EnumTest.Slot.s2

enumTest = Module1.CorbaTests.EnumTest("EnumTest")

retEnumOp enumTest using aSlot = s2 returning s1

```

## enum as the Discriminator of a Union

```
enumTest = Module1.CorbaTests.EnumTest("EnumTest")

unionTest = Module1.CorbaTests.EnumTest.UnionTest()

unionTest.longMember = 10

unionOp enumTest using aUnionTest = unionTest

unionTest.stringMember = "aString"

unionOp enumTest using aUnionTest = unionTest

unionTest.booleanMember = true

unionOp enumTest using aUnionTest = unionTest
```

## CORBA Sequence Examples

Sequences are bound in Methods. If the sequence length exceeds its bound, an exception is thrown. For example, in the following Method script the sequence has 11 members but can only have 10 members.

```
seqTest = Module1.CorbaTests.SeqTest("SeqTest")
stringSeq = [ "1", "2", "3", "4", "5", "6", "7", "8",
              "9", "10", "11" ]

boundStringSeqOp seqTest using
aBoundStringSeq = stringSeq returning stringSeq

display stringSeq
```

When this example is run, it displays the following exception on the screen:

```
Exception:
Sequence size is incorrect, it is 11 and should have been 10.

line=5
column=1
```

## Sequences of primitive types

Any of the primitive types shown in Primitive types can be used to create a sequence. The following example shows how to create a **string** sequence.

```
seqTest = Module1.CorbaTests.SeqTest("SeqTest")

stringSeq = ["one", "two", "three"]

stringSeqOp seqTest using aStringSeq = stringSeq

returning stringSeq

display stringSeq
```

## Sequences of Structs

The Method script below shows how structure sequences can be used in the Method.

```
ts = 'now' // timestamp

seqTest = Module1.CorbaTests.SeqTest("SeqTest")

// create the first struct
structOne = Module1.CorbaTests.SeqTest.TestStruct()
structOne.longMember = 10
structOne.stringMember = String(ts)

// create the second struct
structTwo = Module1.CorbaTests.SeqTest.TestStruct()
structTwo.longMember = 20
structTwo.stringMember = String(ts)

// the array of structures
structSeq = [structOne, structTwo]

testStructSeqOp seqTest using

aTestStructSeq = structSeq
```

## Sequence of Unions

Using sequences of unions is the same as using sequences of structs, as shown in this example:

```
ts = 'now' // timestamp

seqTest = Module1.CorbaTests.SeqTest("SeqTest")

unionOne = Module1.CorbaTests.SeqTest.TestUnion()
unionOne.longMember = 10

unionTwo = Module1.CorbaTests.SeqTest.TestUnion()
unionTwo.stringMember = String(ts)

unionSeq = [unionOne, unionTwo]

testUnionSeqOp seqTest using

aTestUnionSeq = unionSeq
```

## Sequences as INOUT/OUT parameters

Sequences can also be passed as OUT or INOUT arguments in operation invocations. Note that when using sequences of structures or unions as OUT arguments, the restriction that these objects must be instantiated first before being used is still applicable, shown as follows:

```
seqTest = Module1.CorbaTests.SeqTest("SeqTest")

// union is instantiated
testUnion = Module1.CorbaTests.SeqTest.TestUnion()
testUnion.stringMember = "aString"

testUnionSeqOutOp seqTest returning

// union will be received in the sequence
```

```
testUnionSeq = aTestUnionSeq

unionOne = testUnionSeq[0]
unionTwo = testUnionSeq[1]
```

## EJB Components

You can catalog EJB (Enterprise JavaBeans) components and use them in your BPM project.

### Cataloging an EJB Component

Before using EJB components from processes or BPM Objects, you must include them into the project catalog.

Before performing this procedure, you must [Creating an External Resource](#) on page 199 of type *J2EE Application Server* on page 213, with the connectivity information to your JEE container.

In addition, ensure you have created a module where you want to catalog the components. See [Creating a Module](#) on page 152.

1. Right-click on the module where you want to catalog the EJB components.
2. Select **Catalog Component** ➤ **EJB**
3. Select one of the following options:

Option	Description
<b>Use existing configuration</b>	Select this option if you have already configured an External Resource of type Enterprise Java Bean.
<b>Create a new configuration</b>	Select this option if you need to configure a new External Resource for the EJB components. See <a href="#">Creating an External Resource</a> on page 199.

4. Click **Next** and specify an External Resource of type *Java Class Library* on page 215 with the .jar libraries containing your EJB Home and Bean interfaces.

Select one of the following options for the Java Class Library resource:

Option	Description
<b>Use existing configuration</b>	Select this option if you have already configured an External Resource of type Java Class Library.
<b>Create a new configuration</b>	Select this option if you need to configure a new External Resource for the EJB classes components. See <a href="#">Creating an External Resource</a> on page 199.

5. Select the the EJB Home and Bean interfaces you want to catalog.
6. Click **Next**.  
The selected components are analyzed for dependencies. Any errors or warnings are listed.
7. Click **Finish**.

For each selected Java type, a new component appears in the Project Navigator.

### Using EJB Components

Follow this guidelines when using EJB components on you BPM projects.

To access your EJB service from PBL you first use standard component `Fuego.Ejb.EJBHome` to locate the Home interface of your EJB. Using the Home interface, you create a new reference to your Bean.

### Example

```
// Obtain generic reference to an EJB Home object.
// Note: "calculator_ejb" is the name of the External Resource
// defined in your project for "CalculatorHome"
```

```

home as Any = EJBHome.locate(configuration : "calculator_ejb")

// Cast
calculatorHome as CalculatorHome = CalculatorHome(home)

// Create reference to EJB object
calculator = create(calculatorHome)

// Use the object
display "5+5=" + makeSum(calculator, arg1 : 5, arg2 : 5)

// dispose the EJB object
calculator.remove();

```

## JNDI Components

You can catalog directory services using JNDI (Java Naming and Directory Interface) to query, retrieve, delete, and add entries.

### Cataloging JNDI Components

Before performing this procedure, you should ensure that your directory service is configured and running.

Also ensure that you have created a module where you want to catalog the JNDI components. See [Creating a Module](#) on page 152.

To catalog a JNDI Component:

1. Right-click on the module where you want to catalog the JNDI component.
2. Select **Catalog Component ► JNDI**
3. Select one of the following options:

Option	Description
<b>Use existing configuration</b>	Select this option if you have already configured an External Resource of type JNDI Directory Service.
<b>Create a new configuration</b>	Select this option if you need to configure a new External Resource.

4. Click **Next**.  
The list of discovered JNDI classes appears.
5. Select the JNDI classes you want to introspect.
6. Click **Next**.
7. Click **Finish**.

The components you chose to catalog appear in the Project Navigator.

## JNDI Examples

### Performing and LDAP Search

The following example shows how to query for and iterate over a set of LDAP entries using a `for each` statement with a `where` clause.

The condition restricts the query to participants named 'Robert' belonging to the 'Documentation' Organizational Unit.

```

for each hp in humanparticipant
  where hp.ou = "Documentation" and hp.cn = "Robert" do

```

```

        display "The participant: " + hp.cn
    end

```

### Searching LDAP by DN

To retrieve a directory entry by its dn (Distinguished Name), you should use the `lookup( )` method instead of the `for each` statement.

```

myDN = "c=Argentina"
result = lookup(country, dn : myDN)

if (result) then
    delete country
end

```

### Add new LDAP entry

This example shows how to add a new entry to the directory.

```

ol = OrganizationalUnit()
ol.ou = "test21"
ol.description = "foo1"
ol.dn = "ou=test21"
ol.postalCode = "1001"
ol.store()

```

## Java Components

You can catalog Java class libraries (.jar files) and use them on your BPM project. Use Java components to integrate with external APIs or leverage Java code from your BPM project.

### Integrating Java libraries

To catalog Java class libraries you must provide a the .jar files containing the Java classes and their dependencies.

Once the library is cataloged, Studio stores a copy of the .jar files in the `lib/` directory of the project.

### Cataloging Requirements

You can catalog any Java class or interface defined within a named Java package. You cannot catalog Java types defined without a package name.

You must also catalog any additional Java types referred by the Java classes and methods you need to use from your BPM project.

### Runtime Requirements

If the Java classes you catalog depend on other .jar libraries on runtime, you must include these libraries into your BPM project.

All Java code used from a BPM project runs under the control of the [SecurityManager](#) of the Process Execution Engine runtime environment. Its security policies prevent cataloged Java code from performing some operations, including the following:

- Shutting down the JVM
- Spawning Threads or modifying/stopping existing Threads
- Creating new ClassLoaders
- Changing the Socket Factory
- Changing the standard input/output of the JVM

## Cataloging Java Libraries

Before using external Java components from processes or BPM Objects, you must include them into the project catalog. When you catalog a component, you are gathering all the necessary information that Studio needs in order to call and execute it at runtime.

Before performing the procedures in this task, ensure that you have created a module where you want to catalog the Java components. See [Creating a Module](#) on page 152.

To catalog Java components:

1. Right-click on the module where you want to catalog the components.
2. Select **Catalog Component ► Java**
3. Select one of the following options:

Option	Description
<b>Use existing configuration</b>	Select this option if you have already configured an External Resource of type Java Class Library.
<b>Create a new configuration</b>	Select this option if you need to configure a new External Resource of type Java Class Library. See <a href="#">Creating an External Resource</a> on page 199.

4. Select the Java types you want to catalog. Selecting a package automatically selects all types in that package and nested packages.
5. Click **Next**.  
The selected Java types are analyzed for dependencies. Any errors or warnings are listed.
6. Click **Finish**.

For each selected Java type, a new component appears in the Project Navigator.

## Using Java Components

Follow this guidelines when using Java components on you BPM projects.

### Java objects as Process Instance Variables

To use a Java component as a process instance variable, the Java class of the component must be Serializable (it must implement interface `java.io.Serializable`).



**Important:** When the Engine stores a Java component as a process instance variable, it serializes the whole graph of objects referenced by that variable. The bigger the object graph, the more overhead it generates at runtime.

### Versioning Java objects

If your process uses Java components as process instance variables you must be aware of Java class compatibility.

When deploying a new revision of a BPM project, you must ensure that any changes you made to those Java classes do not break compatibility with the previous version of the class.



**Important:** If a new *revision* of a project contains a Java class with an incompatible change, the Process Execution Engine cannot de-serialize process instances holding objects based on older versions of the class.

If you are making incompatible changes to a Java class used as process instance variable you must ensure the following:

- The Java library containing the class is cataloged as **versionable**. See [Versionable Java Libraries](#) on page 169 for details.



- At project deployment time you force a new *version* of your BPM project (instead of a *revision*). When forcing a new project version, the Engine keeps running the old process instances with the old version of the Java class, and new process instances start using the new version of the Java class.

Refer to [Versioning of Serializable Objects](#) of the Java Serialization Specification for details about what changes make a Java class incompatible.

### Versionable Java Libraries

External Resources of type Java Class Libraries can be defined as **versionable** or **non-versionable** to define how the Engine handles the library at runtime.

#### Versionable libraries

Versionable libraries are those that may change as processes evolve. Consequently, it is required that each version of the process to be tied to a specific version of the library. Libraries should be tagged as versionable when the goal is to prevent an update to the library from affecting the behavior of an old version of the process or of a different process that depends on the same components. They could be thought of as integral pieces of a project in the same way the component catalog or processes are.

In general, all new Java classes you write specifically for your BPM project should be tagged as versionable.

In particular, all Java types used as process instance variables should be set as versionable to avoid potential compatibility problems at runtime.

#### Non-versionable libraries

Non-versionable libraries, are those that don't typically change over time. Or, if they do, the intention is for them to affect all versions of all processes deployed in the Engine.

In general, those infrastructure Java libraries or extensions to the BPM system needed to support the execution of processes should be tagged as non-versionable.

For example, libraries for third-party JDBC drivers should be tagged as non-versionable.

### Mapping Java to BPM Components

This section explains how Java types are presented as PBL types when cataloged.

#### Packages to Module mapping

For each Java package you catalog, BPM Studio creates a new module in your component catalog.

Module names are capitalized to follow PBL naming conventions for modules.

#### Java to PBL types

Java types are mapped to PBL types as defined in the following table:

Java Type	PBL Type
java.lang.String, java.lang.Character, char	String
java.lang.Byte, java.lang.Short, java.lang.Integer, java.lang.Long, java.lang.Number, byte, short, int, long	Int
java.math.BigDecimal	Decimal
java.lang.Float, java.lang.Double, float, double	Real
java.lang.Boolean, boolean	Bool
java.util.Date, java.sql.Date, java.sql.Time, java.sql.Timestamp, fuego.lang.Time	Time
byte[]	Binary

Java Type	PBL Type
primitive Java arrays, any implementation of java.util.List, any implementation of java.util.Set, java.util.Vector, any implementation of java.util.Collection.	array
any implementation of java.util.Map or java.util.SortedMap.	associative array
fuego.lang.Interval	Interval

### Oracle Service Bus Components

You can catalog bus services and use them in your BPM project. Oracle Service Bus (formerly AquaLogic Service Bus or ALSB) is a repository for business services exposed as web services.

Oracle Service Bus is an enterprise service bus designed for connecting, mediating, and managing interactions between heterogeneous services. Oracle SB can handle Web services, Java, .Net, messaging services, and legacy end points. Oracle SB is designed to handle the deployment, management, and governance tasks required to implement SOA (Service Oriented Architecture) at any scale.

Oracle Service Bus is stateless and policy-driven. It enables you to establish loose coupling between service clients and business services while maintaining a centralized point of security control and monitoring.

### Supported environments

Oracle BPM 10.3 only supports Oracle Service Bus 10.x (formerly AquaLogic Service Bus 3.0).

Oracle BPM supports the following types of Proxy Service Transports with Service Bus:

- HTTP
- HTTPS
- Oracle BPM (native) Transport

### Cataloging Oracle Service Bus Components

Before performing this procedure, ensure that you have installed and configured Oracle Service Bus (formerly AquaLogic Service Bus or ALSB) and that it is currently running.

Also ensure that you have created a module where you want to catalog the Oracle SB components. See [Creating a Module](#) on page 152.

To catalog a Oracle Service Bus Component:

1. Right-click on the module where you want to catalog the Oracle SB components.
2. Select **Catalog Component ► Oracle Service Bus**
3. Select one of the following options:

Option	Description
<b>Use existing configuration</b>	Select this option if you have already configured an External Resource of type Oracle Service Bus / Management Host.
<b>Create a new configuration</b>	Select this option if you need to configure a new External Resource of type Oracle Service Bus / Management Host. See <a href="#">Creating an External Resource</a> on page 199.

4. Click **Next**.  
Oracle BPM Studio connects to the Oracle Service Bus. This may take several minutes.
5. Select the Oracle SB Project you want to catalog.

Oracle Service Bus resources are organized into projects. You must select a Project that has at least one Proxy Service defined.

6. Select the Proxy Service you wish to use.  
Only projects using the supported transports are shown.
7. Provide a Module name.  
This is the name that will appear in the Project Navigator. The default Module name is the name of the Proxy Service you select.
8. Oracle BPM Studio generates a catalog component for each of the Proxy Service elements.
9. Click **Finish**.

The introspected Oracle Service Bus module appears in the Project Navigator.

### Oracle Service Bus Example

The following example shows how to access Oracle Service Bus objects from within a PBL program.



**Note:** You must catalog Oracle Service Bus objects to the BPM Object Catalog before they can be used within Studio.

### Accessing Oracle SB Objects

The following PBL code shows how to access an Oracle Service Bus object after it has been cataloged. This example is based on the default example that is provided with Oracle Service Bus.

```
service as ServiceBus.LoanGateway1.MyService =
    ServiceBus.LoanGateway1.MyService();
request as ServiceBus.LoanGateway1.LoanStruct =
    ServiceBus.LoanGateway1.LoanStruct();

request.amount = 12345
request.name = 'John Smith'

//display request.name
processLoanApp service
    using loanRequest = request
    returning result2 = result

display result2.name
display result2.notes
```

### SAP Components

You can catalog standard SAP interfaces to integrate your BPM processes with mySAP Business Suite.

Business Application Programming Interfaces (BAPIs) are standard SAP interfaces that enable software vendors to integrate their software into the mySAP Business Suite. BAPIs are implemented using RFC (Remote Function Call) enabled function modules inside SAP systems. BAPIs are defined in the Business Object Repository (BOR) as methods of SAP business objects that perform specific business tasks.

### Requirements

Oracle BPM uses the SAP Java Connector (JCo) to use BAPIs to access SAP. The SAP Java Connector (JCo) is a toolkit that allows Java applications to communicate with SAP systems. JCo is an encapsulation of the RFC Library that supports all features of RFC. Oracle BPM is certified to work with versions 2.0.7 and 2.0.12 of the SAP JCo library.

If you are using the Studio in a Windows environment:

- `librfc32.dll`: Must be located under the `system32` directory.

- `sapjcorfc.dll`: Must be located under the `ext` directory of the Studio installation.

If you are using the Studio in a Unix environment:

- `librfccm.so`: Add to environment variable `LD_LIBRARY_PATH` the full path to the directory containing the file.
- `libsapjcorfc.so`: Must be located under the `ext` directory of the Studio installation.

You must also rename `sapjco.jar` to `sapjco-2.0.jar` and copy it to the following directory of BPM Studio: `<ORABPM_HOME>/studio/eclipse/plugins/fuego.sap_6.5.0/lib/`.

## SAP BAPI Objects, Tables, and Structures

After cataloging SAP BAPI, the following are available:

SAP Object	Description
BAPI Objects	An object is created for each of the introspected BAPIs
SAP Structures	Represent SAP structures which are a set of attributes.
SAP Tables	Represent an SAP table where each row has a set of attributes

## BAPI Objects

Each BAPI Object contains the following:

Imports	Input arguments for the BAPI.
Exports	Output arguments for the BAPI
Tables	Input/Output arguments for the BAPI.
Call Method	Used to invoke BAPI.

## Tables

Use the `currentRow` attribute to access the fields in the current row and the `rows` attribute to get an iterator to access all the rows in a `for` each statement.

## Cataloging SAP Components

Before performing this procedure, ensure you have installed the SAP JCo Library in the SAP Introspector Plugin. See your SAP documentation for more information.

You should also ensure that you have created a module where you want to catalog the components. See [Creating a Module](#) on page 152.

To catalog an SAP BAPI Component:

1. Right-click on the module where you want to catalog the SAP component.
2. Select **Catalog Component ► SAP**
3. Select one of the following options:

Option	Description
<b>Use existing configuration</b>	Select this option if you have already configured an External Resource for the Component.
<b>Create a new configuration</b>	Select this option if you need to configure a new External Resource for the Component. See <a href="#">Creating an External Resource</a> on page 199.

4. Click **Next**.

5. Browse to the location of your SAP JCo file (sapjco.jar).

The first time you add a BAPI with Studio, the SAP JCo is required (sapjco.jar file). Browse the file system to the location where the jar file is and click Next. The application needs to be restarted.

6. Click **Next**.

## SAP Example

The following is a template example that uses the introspected BAPI BAPI\_ALM\_ORDERHEAD\_GET\_LIST. Test is the module's name given when introspecting the BAPI.

```
imports = B.Test.BapiAlmOrderheadGetList.Imports()
tables = B.Test.BapiAlmOrderheadGetList.Tables()

//Completing a Table
// currentRow is used to refer to the row to complete
tables.itRanges = B.Test.BapiAlmOrderListtheadRanges()
tables.itRanges.currentRow.fieldName =
    "OPTIONS_FOR_DOC_TYPE"
tables.itRanges.currentRow.sign = "I"
tables.itRanges.currentRow.option = "EQ"
tables.itRanges.currentRow.lowValue = "PM01"

//Adds the row and moves to the next row to complete
appendRow tables.itRanges

tables.itRanges.currentRow.fieldName =
    "OPTIONS_FOR_PLANPLANT"
tables.itRanges.currentRow.sign = "I"
tables.itRanges.currentRow.option = "EQ"
tables.itRanges.currentRow.lowValue = "5300"

appendRow tables.itRanges

tables.itRanges.currentRow.fieldName =
    "OPTIONS_FOR_COMP_CODE"
tables.itRanges.currentRow.sign = "I"
tables.itRanges.currentRow.option = "EQ"
tables.itRanges.currentRow.lowValue = "5560"

appendRow tables.itRanges

tables.itRanges.currentRow.fieldName =
    "SHOW_COMPLETED_DOCUMENTS"
tables.itRanges.currentRow.sign = "I"
tables.itRanges.currentRow.option = "EQ"
tables.itRanges.currentRow.lowValue = "X"

appendRow tables.itRanges

tables.itRanges.currentRow.fieldName =
    "SHOW_DOCUMENTS_IN_PROCESS"
tables.itRanges.currentRow.sign = "I"
tables.itRanges.currentRow.option = "EQ"
tables.itRanges.currentRow.lowValue = ""

appendRow tables.itRanges

tables.itRanges.currentRow.fieldName =
    "SHOW_OPEN_DOCUMENTS"
tables.itRanges.currentRow.sign = "I"
tables.itRanges.currentRow.option = "EQ"
```

```

tables.itRanges.currentRow.lowValue = ""

appendRow tables.itRanges

tables.itRanges.currentRow.fieldName =
    "OPTIONS_FOR_CHANGE_DATE"
tables.itRanges.currentRow.sign = "I"
tables.itRanges.currentRow.option = "GE"
tables.itRanges.currentRow.lowValue = "20060710"

// "Exports" is defined to receive
// the returned result from the BAPI
Exports as B.Test.BapiAlmOrderheadGetList.Exports

// Invoking the BAPI using the "call" method
call B.Test.BapiAlmOrderheadGetList
    using imports,
        tables
    returning tables = tables,
        Exports = Exports

// Table iteration to display the rows
for each row in tables.itRanges.rows do
    display row
end

// Working with a table
if size(tables.etResult) > 0 then
    display tables.etResult
end

```

## SQL Components

You can catalog SQL tables, views and store procedures. Use SQL components from your processes and BPM objects to query, update and execute SQL statements on external databases.

## Supported Databases

You can catalog components from JDBC-compliant databases. You need a special JDBC driver for each database vendor.

Oracle BPM includes JDBC drivers for several database vendors including Oracle, IBM DB2, Microsoft SQL Server, Informix and Sybase. To use an external JDBC driver you must first include the . jar files of the driver as an External Resource of type [Java Class Library](#) on page 215.

## JDBC Connections on Runtime

The Process Execution Engine manages and pools the connections to the database servers automatically.

When the Engine runs on an EJB container (such as Oracle WebLogic or IBM WebSphere), it leverages the JDBC data source functionality provided by the container. In such environments, all JDBC connectivity and connection pooling is handled by the EJB container.

## Cataloging a SQL Component

Before using SQL tables from processes or BPM Objects, you must include them into the component catalog. When you catalog SQL table components, Studio gathers all the information it needs for integrating with your SQL database and executing statements at runtime.

Before performing this procedure, you should ensure that your database is installed, running and accessible from your system through JDBC.

You should also ensure that you have created a module where you want to catalog the SQL components. See [Creating a Module](#) on page 152.

To catalog a SQL Component:

1. Right-click on the module where you want to catalog the SQL component.
2. Select **Catalog Component ► SQL**
3. Select one of the following options:

Option	Description
<b>Use existing configuration</b>	Select this option if you have already configured an External Resource of type SQL Database.
<b>Create a new configuration</b>	Select this option if you need to configure a new External Resource of type SQL Database. See <a href="#">Creating an External Resource</a> on page 199.

4. Click **Next**.  
Oracle BPM Studio creates a list of SQL objects that can be cataloged. This may take several minutes.
5. Select the SQL components you want to catalog.
6. Click **Next**.  
The components are analyzed for dependencies and cataloged.
7. Click **Finish**.

The components you chose to catalog appear in the Project Navigator.

### Using SQL Components

You can catalog SQL tables, views and store procedures. Use SQL components from your processes and BPM objects to query, update and execute SQL statements on external databases.

Oracle BPM provides developers the following mechanisms for accessing JDBC-compliant databases:

- Using cataloged SQL components. For each SQL table, view or store procedure, Studio generates a BPM component with attributes and methods to read and update tables, and call procedures. In general, each BPM component represents a table and each component instance represents a row in that table. SQL components inherit from the standard `Fuego.Sql.SqlObject` component.
- Using SQL syntax embedded in PBL code. This allows you use SQL statements which are statically checked at compilation time. A subset of the standard SQL language is accepted. Vendor-specific statements are not allowed in embedded SQL code. See [Embedded SQL Overview](#) on page 359 for more details.
- Using the standard `Fuego.Sql.DynamicSQL` component. This component gives you the most flexibility but provides no static checks at compilation time. With this component, you can build SQL statements dynamically at runtime, and you can send any command to your database. Refer to the `Fuego.Sql.DynamicSQL` component documentation for more details.

### SQL Component methods

Components for cataloged SQL tables provide three methods:

- `load()`: Loads a row from the database into the component attributes.
- `store()`: Inserts/updates the component attribute values to the database.
- `remove()`: Deletes the database row associated with this component instance.

Refer to the reference documentation of the standard `Fuego.Sql.SqlObject` component for details.

Components for cataloged stored procedures provide a single method:

- `call(...)`: Executes the store procedure. This method accepts the arguments defined by the stored procedure.

## Auto-loading

If the primary key attributes of SQL table component are set, the component automatically calls its `load()` method the first time you read or set one of its non-key attributes. Example:

```
customer = CUSTOMER() // New instance of CUSTOMER table component
customer.id = "1234"   // sets value of primary key attribute

logMessage "Customer Name=" +
    customer.name // this triggers an implicit call to "load()"
```

You can disable this automatic loading behavior by setting the `accessDatabase` attribute of the component to `false`. This example is equivalent to the previous one but without using auto-loading:

```
customer = CUSTOMER() // New instance of CUSTOMER table component
customer.id = "1234"   // sets value of primary key attribute
customer.accessDatabase = false // disable auto-loading

// Note: "customer.name" evaluates to null (row not auto-loaded)

customer.load()        // load row explicitly

logMessage "Customer Name=" + customer.name
```

## Handling Database Connections

The Process Execution Engine manages and pools the connections to the database servers automatically.

From PBL code you don't explicitly open or close JDBC connections. When your PBL code instantiates a SQL component, the Engine automatically assigns a connection to that database from the pool. When all SQL components for that database get out of scope, the Engine releases the connection back to the pool.

SQL Components connecting to the same database within the same transaction share a single connection.

## Handling Database Transactions

The Process Execution Engine always executes PBL code in the context of a transaction. In general, PBL code is called from a process activity, and the execution of the activity task defines the transaction boundaries.

From PBL code you don't explicitly start, commit or rollback a database transaction. When your PBL code first instantiates a SQL component, the Engine automatically starts a transaction on that database.

When the execution of a PBL script succeeds (no Exceptions thrown), the Engine commits all associated database transactions. When the execution of a PBL script fails (an Exception is thrown or predefined variable *action* indicates so) the Engine rolls back all associated database transactions.

When the Process Exception Engine runs on an EJB container (such as Oracle WebLogic or IBM WebSphere) and all JDBC data sources involved are configured to be "XA" (for distributed transactions), then database connections will participate on the same global transaction managed by the container.

## SQL on client-side methods

SQL components can only execute in the context of the Process Execution Engine.

If you are using SQL components from a BPM object method, you should set property **Service Side Method** of your method to **Yes**.

Server-side methods cannot be invoked directly from BPM Object Presentations. If you need to access SQL components from a Presentation, extract the code that uses SQL into a separate method, make this new method **Server Side** and call it from the original client-side method.



## SQL Components as Instance Variables

You can use cataloged SQL table components as process instance variables. This is useful when you want to persist data associated with a process instance into a separate database.

Unlike any other component types, the Process Execution Engine treats process instance variables of a SQL table component type in a special way.

When the Engine completes the execution of a transaction on a process instance (most commonly, the execution of a process activity) the Engine persists the state of all instance variables on its runtime database. However, for those instance variables of a SQL table component type, the Engine only persists its primary key attributes.

When you access the attributes of a SQL table object used as instance variable, the component automatically loads its attributes from the database. Refer to [Using SQL Components](#) on page 175 for details on how SQL components are auto-loaded.

This special behavior for SQL table components provides the following benefits:

- Prevents unnecessary duplication of information. Since you do not keep the information separately from the original database, you always access the latest updated information.
- The SQL component is auto-loaded on demand. The component does not query the database until your code sets or reads one of its attributes.
- Reduces the size of your process instances payload. This not only saves storage space but also improves process execution performance.
- 

## Disabling special handling of SQL components

You can disable this behavior by setting the `accessDatabase` attribute of the component to `false`.

When attribute `accessDatabase` is `false` on a SQL component used as a process instance variable, the Process Execution Engine persists the complete state of the variable, including all primary key and non-primary key attributes.

When attribute `accessDatabase` to `false`, auto-loading is disabled. Refer to [Using SQL Components](#) on page 175 for details on how SQL components are auto-loaded.

```
// "customer" is a process instance variable of
// SQL table type "CUSTOMER"
this.customer.accessDatabase = false

//... from now on, the Engine stores the complete state of
// "customer" variable, including non-primary key fields.
// Auto-loading is also disabled for this variable.
```

## Mapping SQL to BPM Components

This section explains how JDBC/SQL objects and types map to BPM components and PBL types when cataloging SQL tables, views and procedures.

When cataloging SQL objects, Studio creates a new BPM component for each table, view and store procedure you select. These auto-generated components inherit from the standard `Fuego.Sql.SqlObject` component (refer to the *Oracle BPM Components Reference* for details).

## Renamed elements and attributes

The naming conventions on Java and PBL dictate that component names should start with uppercase letters and attribute names should start with lowercase letters. All names should use "camel-case" to separate words and not use underscores ("\_").

When cataloging SQL objects, the attributes of generated components will follow Java conventions and the original SQL names are converted if necessary.

For example, for cataloged SQL column named `purchase_order`, the corresponding BPM component attribute is named `purchaseOrder`.

### JDBC column types to PBL types

JDBC column types are mapped to PBL types as defined in the following table:

JDBC Type	PBL Type
CHAR	String
LONGVARCHAR	String
VARCHAR	String
CLOB	String
TIME	Time
DATE	Time
TIMESTAMP	Time
BIT	Bool
TINYINT	Int(8)
SMALLINT	Int(16)
INTEGER	Int
BIGINT	Int(64)
NUMERIC	Decimal
DECIMAL	Decimal
FLOAT	Real(32)
DOUBLE	Real
REAL	Real
BINARY	Binary
LONGBINARY	Binary
VARBINARY	Binary
BLOB	Binary
OTHER	Java.Lang.Object

Some database provide non standard column types. The following table shows how vendor-specific types map to PBL types:

DB Vendor	Vendor Type	JDBC Type	PBL Type
MS-SQL	NVARCHAR	VARCHAR	String
MS-SQL	NCHAR	CHAR	String
MS-SQL	NTEXT	CHAR	String
DB2	CHARACTER	CHAR	String
Oracle	DATE	DATE or TIMESTAMP	Time

Oracle DATES can store dates with or without the time component. The type mapping depends on the **Use Timestamp for Date columns** option on the External Resource configuration.

## SQL Query Components

You can catalog predefined SQL queries as components in your project catalog.

Studio allows you to create components that encapsulate predefined SQL queries. These SQL queries can also receive arguments and are reusable across the project.

Think of cataloged SQL queries as SQL views defined on the client-side.

Studio generates a component containing one attribute for each column returned by the query. Only SELECT-type queries are allowed and the generated component provides read-only access to the results.

### Cataloging a SQL Query

The following procedures show you how to catalog a table using a SQL Query.

Before performing these procedures, ensure that you have create a Module within the Catalog.

1. In the Project Navigator, right-click on the Module where you want to Catalog a SQL query.
2. Select **Catalog Component ► SQL Query**
3. Select one of the following options:

Option	Description
<b>Use existing configuration</b>	Select this option if you have already configured an External Resource of type SQL Database.
<b>Create a new configuration</b>	Select this option if you need to configure a new External Resource of type SQL Database. See <a href="#">Creating an External Resource</a> on page 199.

4. Click **Next**.
5. Enter a name for the new component that represents your query.
6. Enter the SQL "SELECT" statement to query your database.  
Select the **Parametric** option if your query defines placeholders to be parametrized at runtime. See [Parametric Queries](#) on page 179 for details.
7. If the **Parametric** options is not selected, you may click **Preview** to test your query before continuing.
8. Click **Next**.

The SQL query is executed. This may take a few minutes.

If the query defines parameters, you may now fill in sample values for each parameter and click **Preview** to test your query before continuing, then click **Next**

The list of columns returned by the query are displayed in a table.

9. Click **Next**.

The component is cataloged.

10. Click **Finish**.

The SQL query component appears in the Project Navigator .

### Parametric Queries

When you catalog a SQL query, you have the option of making it parametric. Parametric SQL Queries accept arguments at runtime.

When you define the SQL statement of a parametric query, you can insert special markers in the place of actual values. Studio then generates a component that accepts arguments to fill into the placeholders when invoked.

This allows you specialize the query according to values that are determined at runtime, instead of design time when you design and catalog the query.

Syntax

To use parametric markers in your SQL statement, use the following syntax: \$name : type, where "name" is the name of the parameter and "type" is the SQL type of the parameter value.

Each parameter defined by the query becomes an argument to the constructor of the generated query component.

Example

Example: Here's a simple parametric query for a hypothetical component named EmployeesByDepartment:

```
SELECT * from EMPLOYEE WHERE deptNumber = $dept:VARCHAR
```

When using the query component, you pass the value for the "dept" placeholder to the constructor, as follows:

```
for each hrEmployee in EmployeesByDepartment( dept: "HR1" )
do
    // ...
end
```

Using SQL Query Components

Use the for each statement to execute and iterate over the result set of a cataloged SQL query.

Query components define one attribute for each column returned by the query.

Simple Queries

To execute the query, simply iterate over the component using the for each statement:

```
for each record in TotalItemsQty do
    display record.qty
end
```

Parametric queries

To execute a parametric query, you must provide the values for all its parameters to the constructor of the query component. The following example passes values for the "type" parameter defined by a query:

```
for each record in TotalItemsQtybyType(type: "myType") do
    display record.qty
end
```

Mapping SQL types to PBL

This sections explains how JDBC/SQL types map to PBL types when cataloging SQL queries.

JDBC column types to PBL types

JDBC column types are mapped to PBL types as defined in the following table:

JDBC Type	PBL Type
CHAR	String
LONGVARCHAR	String
VARCHAR	String
CLOB	String
TIME	Time
DATE	Time

JDBC Type	PBL Type
TIMESTAMP	Time
BIT	Bool
TINYINT	Int(8)
SMALLINT	Int(16)
INTEGER	Int
BIGINT	Int(64)
NUMERIC	Decimal
DECIMAL	Decimal
FLOAT	Real(32)
DOUBLE	Real
REAL	Real
BINARY	Binary
LONGBINARY	Binary
VARBINARY	Binary
BLOB	Binary
OTHER	Java.Lang.Object

Some database provide non standard column types. The following table shows how vendor-specific types map to PBL types:

DB Vendor	Vendor Type	JDBC Type	PBL Type
MS-SQL	NVARCHAR	VARCHAR	String
MS-SQL	NCHAR	CHAR	String
MS-SQL	NTEXT	CHAR	String
DB2	CHARACTER	CHAR	String
Oracle	DATE	DATE or TIMESTAMP	Time

Oracle DATES can store dates with or without the time component. The type mapping depends on the **Use Timestamp for Date columns** option on the External Resource configuration.

### Web Service Components

You can catalog SOAP Web Services from a WSDL (Web Services Description Language) definition and use them in your BPM project. Use Web Service components to integrate with external systems using the SOAP standard.

When you catalog a SOAP service you get a BPM component which provides one method for each SOAP operation exposed by the service. If the web service defines complex data types, additional BPM components are generated to represent them.

### Supported Standards

Oracle BPM supports WSDL version 1.1 (Web Services Description Language) and SOAP versions 1.1 and 1.2.

Oracle BPM supports sending and receiving SOAP messages using RPC (Remote Procedure Call) and Document styles.

### Supported Transports

Oracle BPM supports the following transports for consuming SOAP services:

<b>HTTP</b>	One-way and Request-Response interactions.
<b>JMS</b>	One-way interactions over Java Messaging Service.
<b>Oracle BPM native</b>	One-way and Request-Response interactions. For integration with Oracle Service Bus.

### Cataloging Web Service Components

Before performing this procedure ensure you have created a module where you want to catalog the Web Service components. See [Creating a Module](#) on page 152.

To catalog Web Service Components:

1. Right-click on the module where you want to catalog the components.
2. Select **Catalog Component ► Web Service**
3. Select one of the following options:

Option	Description
<b>WSDL file</b>	Select this option to specify the location to the WSDL file. Fill in the <b>WSDL Address</b> field with an HTTP URL or a file path on your local machine.
<b>UDDI inquiry</b>	Select this option to access the WSDL definition through a UDDI (Universal Description Discovery and Integration) registry. Fill in the <b>Inquiry URL</b> field with the URL to your UDDI registry. You may specify authentication credentials if your registry requires it. Enable option <b>UDDI Dynamic Endpoint Binding</b> to query the UDDI registry and resolve the endpoint address at runtime.

4. Provide a name for the new sub-module in the **Module** field.
5. Review all the information and click **Next**.
6. If you selected to use UDDI inquiry:
  - a) Enter the type (Business or Service) and name to search for.
  - b) Click Next
  - c) If you searched for Businesses, select one of the businesses listed and click Next.
  - d) Select one of the Services from the list and click Next.
  - e) Select one of the endpoints listed and click Next.
7. Oracle BPM Studio generates a catalog component for each of the Web Service elements.
8. Click **Finish**.

The cataloged Web Service module appears in the Project Navigator.

### Using Web Service Components

Use Web Service components to invoke external SOAP services.

### Invoking a cataloged Service

Use the `*Service` interface to invoke operations on a cataloged Web Service. For example, the following code uses a hypothetical Currency Exchange service:

```
// Instantiate a Service component
// Configuration taken from associated External Resource
exchange as CurrencyExchangeService = CurrencyExchangeService()
```

```
// Invoke the "getRate" operation
rate = exchange.getRate(from: "USD", to: "EUR")
```

## Overriding Endpoint configuration

If you use the default constructor of a `*Service` component, it reads the configuration values from the associated External Resource.

You can override the External Resource configuration passing a String with the URL of the desired endpoint, or passing an object of type `Fuego.WebServices.Configuration`. For example, the following code uses the hypothetical Currency Exchange service, explicitly defining the endpoint information:

```
// Build new WebService configuration:
wsConfig = Fuego.WebServices.Configuration()
exchangeEndPoint =
HttpEndpoint("http://localhost:8080/webservices/CurrencyExchange")
wsConfig.endpoint = exchangeEndPoint

// Instantiate a Service component, passing a configuration
// object and ignoring the associated External Resource
exchange as CurrencyExchangeService = CurrencyExchangeService(wsConfig)

// Invoke the "getRate" operation
rate = exchange.getRate(from: "USD", to: "EUR")
```

Refer to the documentation of `Fuego.WebServices.Configuration` component for more details.

## Handling Exceptions

When a cataloged Web Service component encounters a SOAP fault, it throws a `Fuego.WebServices.S SoapFaultException` that contains the SOAP Fault code and description string.



**Important:** This exception only covers faults generated by the SOAP protocol. Lower-level network problems and exceptions (like `ConnectionException` or `NoRouteToHostException`) are not covered by `SoapFaultException`.

For example, the following code uses the hypothetical Currency Exchange service, explicitly handling exceptions caused by the component:

```
do

// Invoke service
exchange as CurrencyExchangeService = CurrencyExchangeService()
rate = exchange.getRate(from: "USD", to: "EUR")

on soapFault as SoapFaultException

    logMessage "SOAP fault caught: [" + soapFault.faultCode + "] "
        + soapFault.faultString

    logMessage "Fault message:" + generateXmlFor(soapFault)

    // ...
on ex as Exception

    logMessage "Non-SOAP exception calling Web Service " + ex
    // ...
```

```
end
```

### Mapping SOAP Web Services to BPM Components

This section explains how SOAP services map to BPM components when cataloging WSDL definitions.

When cataloging a WSDL descriptor, Studio creates a new BPM component for each service the WSDL defines. The name of the component ends with `Service`. The component provides methods for each operation exposed by the service.

### Complex types

Since WSDL uses XML Schema (XSD) to describe complex data types, Studio maps all types defined in the WSDL descriptor to XSD components as described in [Mapping XSD to BPM Components](#) on page 188. For each complex XSD type, Studio generates a new BPM Component.

Port types and Messages defined in the WSDL also map to XSD components.

Fault Messages in the WSDL map to `Fuego.WebServices.SoaFaultException` in the catalog.

### XML Schema Components

You can catalog XML Schema Definitions (XSD) components and use them in your BPM project. Use XML Schema components for parsing, validating, manipulating and generating typed XML documents.

### Supported XML Schema languages

Oracle BPM supports the XML Schema Definition (XSD) language versions 1.0 and 1.1.

Other XML schema languages, such as like DTD (Document Type Definition) are not currently supported. Third party tools exist to convert other schema languages into XSD.

### Not using Schemas

If you need to perform ad-hoc XML parsing and general XML processing without using schemas, refer to the standard `Fuego.Xml.XMLDocument` and `Fuego.Xml.XMLNode` components.

### Cataloging XML Schema

Before using XML Schema components from processes or BPM Objects, you must include them into the project catalog. When you catalog a component, you are gathering all the necessary information that Studio needs in order to call and execute it at runtime.

Before performing the procedures in this task, ensure that you have created a module where you want to catalog the XML Schema components. See [Creating a Module](#) on page 152.

To catalog an XML Schema Component:

1. Right-click on the module where you want to catalog the component.
2. Select **Catalog Component ► XML Schema**
3. Enter the location of the schema file you want to catalog.



**Note:** Only XML Schema Definitions (.xsd) can be cataloged.

4. Enter a Module Name
5. Click **Next**.  
The XML schema is analyzed for dependencies. Any errors or warnings are listed.
6. Click **Finish**.

For each type defined in the XML Schema Definition, a new component appears in the Project Navigator.

### Using XML Schema Components

Use XML Schema components for parsing, validating, manipulating and generating typed XML documents.



## Loading XML documents

To create a typed XML object from an XML string use method `load()`.

To create a typed XML object from an external XML file use method `loadFromUrl()`. This method supports `file://` and `http://` protocols.

You can also work with the `Fuego.Xml.XMLObject` component directly, although it doesn't provide any type information or validation.

Loading malformed documents causes a runtime Exception.

## Validating an XSD component

When an XSD component loads an XML document (methods `load`, `loadFromUrl`) it does **not** validate the document against the schema. The component ignores unknown XML elements and attributes when loading, but these are not discarded: generating an XML string representation (with `generateXmlFor()`) includes the original elements and attributes.

To validate a loaded document, use the `validate()` method. If validation fails, this method raises a runtime Exception.

## Converting XSD components to XML strings

All cataloged XSD components provide method `generateXmlFor` (inherited from `Fuego.Xml.XMLObject`). This method serializes the current `XMLObject` to an XML String representation according to the XML-Schema 1.1 specification.

## XSD Components on runtime

The XML Schema Definition documents (\*.xsd) are not needed on runtime. Oracle BPM does not use the XSD file once its been cataloged.

### XML Schema Examples

An example of using XML Schema Definition representing customer information.

### XML Schema Introspection Example

The following is an XML schema example for an XML document that may be used in a business process.

```
<?xml version="1.0" encoding="UTF-8" ?>

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:simpleType name="Address" type="xsd:string"/>
  <xsd:simpleType name="CustomerCode" type="xsd:string"/>
  <xsd:simpleType name="CustomerName" type="xsd:string"/>
  <xsd:simpleType name="CodPay" type="xsd:string"/>
  <xsd:simpleType name="CustDisc" type="xsd:string"/>
  <xsd:simpleType name="CustType" type="xsd:string"/>
  <xsd:simpleType name="Mail" type="xsd:string"/>

  <xsd:group name="shipAndBill">
    <xsd:sequence>
      <xsd:element name="ShipAddress" type="Address"/>
      <xsd:element name="BillAddress" type="Address"/>
    </xsd:sequence>
  </xsd:group>

  <xsd:complexType name="Customer">
    <xsd:sequence>
      <xsd:choice>
        <xsd:element name="CustomerCode"
          type="CustomerCode"/>

```

```

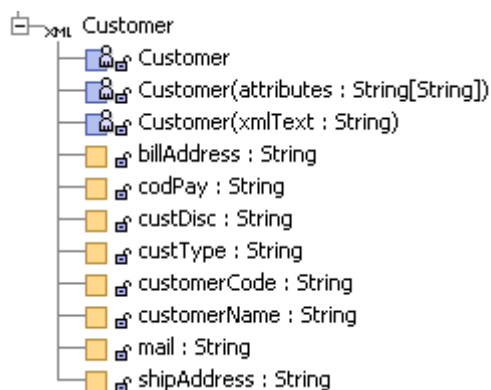
        <xsd:element name="CustomerName"
            type="CustomerName"/>
        <xsd:element name="CodPay" type="CodPay"/>
        <xsd:element name="CustDisc" type="CustDisc"/>
        <xsd:element name="CustType" type="CustType"/>
        <xsd:group ref="shipAndBill"/>
        <xsd:element name="Mail" type="Mail"/>
    </xsd:choice>
</xsd:sequence>
</xsd:complexType>


<xsd:element name="customer" type="Customer"/>

</xsd:schema>

```

After cataloging the XML Schema Definition detailed above, the Project catalog structure appears as shown in the following image:



 **Note:** You can drag and drop the XML attributes and methods to the method editor in order to visualize the usage template.

### Create an XML File Based on an the cataloged components

```

customer as XMLComp.Customer.Customer
customer = XMLComp.Customer.Customer()

customer.customerCode = "NEW"
customer.customerName = "John Smith"
customer.custDisc = null
customer.billAddress = "Madison 2939 NY"
customer.shipAddress = "Madison 2939 NY"
customer.codPay = null
customer.custType = null
customer.mail = "pp@com.com"

store customer
    using targetFile = "C:/tmp/customer.xml"

```

## Loading an XML File Using loadFromUrl

The `loadFromUrl()` method can receive as parameter an URL pointing to the xml file, like `file://home/sharedDocuments/test.xml` or `http://...`

```
customer as XMLComp.Customer.Customer
customer = XMLComp.Customer.Customer()

customer.loadFromUrl("file://c:/tmp/customer.xml")
```

The `loadFromUrl()` method uses DOM (Document Object Model) to load the complete XML document in memory.

## Load an XML data file containing a customer using the load method

The **load** method can receive as parameter:

- an URL
- a path location to the XML file. For example: `C:\\documents\\test.xml` or `/documents/test.xml`
- the XML document content.

## Important Considerations

Advanced information you may need to know when working with cataloged XML Schema components.

### About <xsd:import> and <xsd:include>

When an XSD documents uses `<xsd:import>` or `<xsd:include>` definitions, the referred XSD documents will also be loaded and cataloged together in the same module.

If you catalog two separate XSD documents which both include a common third XSD document, the types defined in the shared XSD will be included twice in your catalog.

For example, given the following scenario:

```
Customer.xsd includes Address.xsd
Provider.xsd includes Address.xsd
```

If you catalog `Customer.xsd` you will get components for both **Customer** types and **Address** types under the **Customer** module. If you then catalog `Provider.xsd` you will get components for both **Provider** and (again) **Address** types under the new **Provider** module. Those types defined in the shared `Address.xsd` appear twice.

Although **Address** components under the **Customer** and **Provider** modules are not strictly of the same PBL type, you can convert objects from one to the other by converting them to XML strings and re-creating the objects. Example:

```
customer as XSD.Customer.Customer
customer = XSD.Customer.Customer()
customer.loadFromUrl("file://c:/tmp/customer.xml")

// customer.address is of PBL type XSD.Customer.Address

xmlAddress as String = generateXmlFor(customer.address)

provider as XSD.Provider.Provider()
provider.address = XSD.Provider.Address(xmlAddress)
```

**Mapping XSD to BPM Components**

This sections explains how XML Schema Definition (XSD) types map to BPM components and PBL types when cataloging XSD components.

The document structures defined in the XSD are converted to regular BPM components.

For each complex XSD type, a new BPM Component is generated. XML elements and XML attributes become attributes of the BPM components. All generated XSD components inherit from the standard `Fuego.Xml.XMLObject` component (refer to the *Oracle BPM Components Reference*).

If the type of an XML element is complex, then the type of the associated component attribute will be of that of the component created for that complex type. For example, if a complex `<customer>` element includes a complex `<address>` element, you get two BPM components (Customer and Address) and the Customer component contains an attribute named `address` of type Address.

**Renamed elements and attributes**

The naming conventions on Java and PBL dictate that component names should start with uppercase letters and attribute names should start with lowercase letters. All names should use "camel-case" to separate words and not use underscores ("\_").

When cataloging XSDs, the generated components' attributes will follow Java conventions and the original XML element and attribute names will be converted if necessary.

For example, if an XSD defines an type named `purchase_order`, the corresponding BPM component is named `PurchaseOrder`.

**Simple XSD to PBL types**

Simple XML types are mapped to simple PBL as defined in the following table:

XSD Type	PBL Type
base64Binary	Binary
hexBinary	Binary
anyURI	String
anguage	String
normalizedString	String
string	String
token	String
byte	Int(8)
decimal	Decimal
double	Real
float	Real(32)
int	Int
integer	int(64)
long	int(64)
negativeInteger	Int(64)
nonNegativeInteger	Int(64)
nonPositiveInteger	Int(64)
positiveInteger	Int(64)

XSD Type	PBL Type
short	Int(16)
unsignedByte	Int(8)
unsignedInt	Int
unsignedLong	Int(64)
unsignedShort	Int(16)
date	Time
dateTime	Time
duration	Interval
gDay	Int
gMonth	Int
gMonthDay	Time
gYear	Int(16)
gYearMonth	Time
time	Time
anyType	Fuego.Xml.XMLObject
anySimpleType	String
Name	String
NCName	String
NOTATION	String
QName	String
ENTITY	String
ENTITIES	String[]
ID	String
IDREF	String
IDREFS	String[]
NMTOKEN	String
NMTOKENS	String[]

## BPM Objects

### BPM Object Overview

A BPM Object is a user-defined component that contains *attributes*, *methods*, and *presentations*.

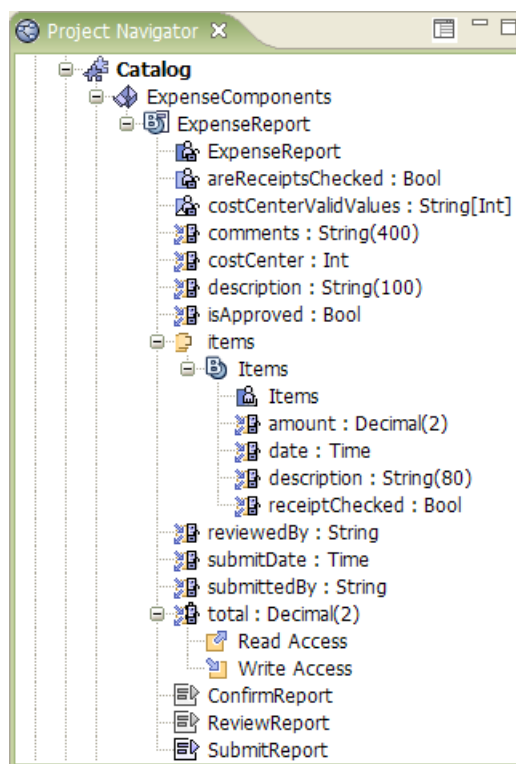
A BPM Object can be used to encapsulate any type of information that the process requires. For example, It can be to input and store information that would be persisted in another type of data container, such as a database or an XML file.

A BPM Object is composed of:

BPM Object Element	Description
Attributes	<p>Attributes are data elements (like variables), used to store data that define and describe the BPM Object.</p> <p>Attributes can be defined as virtual, in which case they do not actually contain any data, but are instead implemented as a pair of methods for reading and writing. Virtual attributes are accessed like regular attributes.</p>
Methods	<p>Methods are like functions or subroutines associated to the object. You write methods in Process Business Language, and can use them to access or set BPM Object data indirectly. For example, you may want to obtain the sum of several numeric attributes. In this case you can read each attribute from the BPM Object and add them, or you can add a method to the BPM Object, called for example <code>attributeSum</code>, which will return the summed value.</p> <p>The resulting code is easier to maintain. For example, if you add a new attribute to the BPM Object which must also be summed, you can edit the <code>attributeSum</code> method to include the new attribute. By doing it this way, you avoid the need to track every piece of code which requires the sum of the attributes of the object.</p>
Groups	<p>Groups are objects made up of one or more related attributes and stored in an array. Groups are designed to be used wherever you require a list of items and each item has several attributes. For example, in an invoice there can be several items, and each item has a description, a quantity, and a price.</p> <p>If you are a Java programmer, you can think of a group as being analogous to an inner class.</p>
Presentations	<p>BPM Object Presentations are essentially forms which either show or allow input of BPM Object properties. A presentation can show all or some of the properties of a BPM Object</p> <p>Every BPM Object may contain one or more presentations. Each of the fields on the presentation are tied to one of the attributes. Presentations provide a simple way for end users to view or to input the attributes of the BPM Object.</p> <p>When a BPM Object is created, it is only a data container or non-presentable BPM Object until a presentation is added to it; then it becomes presentable.</p> <p>See <a href="#">End-User Interfaces on Oracle BPM</a> on page 250 for more information about how BPM Object data is presented to end users.</p>

### Example BPM Object

The following figure shows the structure of an expense report BPM Object:




**Figure 4: Expense Report BPM Object**

The BPM Object above shows methods, attributes, groups, and presentations. Most of these were added by the developer, while a few of the methods were created automatically based on the properties set when creating the different attributes. An example of an automatically generated method is the `costCenterValidValues` method. This method is created by setting a list of valid values to the `costCenter` attribute when defining it. This method returns an array containing all the values allowed for that attribute, which is used by presentations to display a drop-down list with the possible values of an attribute.


## Creating a BPM Object

You define BPM Objects in several steps. The first step is to create the BPM Object, where you will specify its basic properties. Afterwards, you can add attributes, groups, methods, and presentations.

 **Note:** If you are familiar with object-oriented programming, you can think of creating a BPM Object as defining a class.

To create a BPM object:

1. In the **Project Navigator**, within your project, expand **Catalog** (🔗).  
You will see a list of catalog modules. If you are working on a new project, these include: *Fuego*, *Java*
2. You define BPM Objects in your own modules. To add your own module, right-click on Catalog and click **New ► Module** (🔗).

 **Note:** You can define several BPM Objects in one module, so you do not need to execute this step if you already have defined a module.

The **Module** dialog box appears.

3. Enter a name such as `MyModule` in the **Module Name** field, and click OK.  
The module you specified is added to the catalog.

4. Right-click on the module icon and click **New ► BPM Object** (B).  
The **BPM Object** dialog box appears.
5. Enter a name for the new object, such as `MyObject`, in the **Name** field. Click **OK**.  
The BPM object is added to the module.
6. Expand your module and then expand your new BPM object.  
You will see the contents of the new object. It contains one *method*, with the same name as the BPM Object. This is the *constructor* method, which means that it will execute whenever an object of this type is created (or "constructed"). If you need to include code that will initialize something in the BPM object, you can add it to this method.

## Attribute Overview

Attributes are the data elements, such as numbers, strings, or date values, that describe the state and contents of a BPM Object.

### Read and Write Access

If an attribute has no read access, this means that its value is not accessible. On the contrary, if an attribute has no write access, its value cannot be changed.

All real attributes have both read and write access set by default. When an attribute is added to a BPM Object, it is created as a real attribute by default. When access methods are added to real type attributes, they override this default. When access methods are removed, they reset to the default. These changes never modify the read/write access of the attribute.

An attribute can be redefined as real or virtual by selecting or deselecting the Virtual check box available for the attribute in the BPM Object editor panel.

You must define at least one access method for a virtual attribute. If no access method is defined, the BPM Object compilation fails.

Read access has a return type identical to the return type of the attribute. Write access receives an argument called `value`, which is of the same type and has no return type.

### When to Use a Real or Virtual Attribute

You must use a real attribute when you need to store a data element. You should use virtual attributes to expose values which can be calculated from existing real attributes. It may not always be obvious which values should be calculated and which should be directly stored. In the temperature example above we could have stored the Fahrenheit temperature while calculating the Celsius temperature

As a general rule, you should store what you consider to be the most natural value or the value you expect to use the most. The main thing is to avoid storing redundant information, since you risk ending up with divergent values for the same thing. Continuing with our example, if the object stores the temperature in Fahrenheit and also in Celsius, it will be possible to change only one of the values. The BPM Object would then contain two divergent values.

## Defining an Attribute

You define an attribute in Studio from the Project Navigator.

To define a BPM Object attribute:

1. In the Project Navigator, right-click on the BPM Object (B), and click **New ► Attribute** (A).  
The **Attribute** dialog box appears.
2. Enter the attribute name in the **Name** field.
3. Select the data type from the **Type** drop-down list, and click **OK**.  
The attribute editor for the new attribute opens.



4. For some data types, you can specify length or precision, as shown below:

Data Type	Property	Possible values
String	Maximum Length	1 to 10,000
Decimal	Decimal Digits	0 to 100
Time	Time Precision	Date Only, Time Only, or Timestamp

5. In the **Storage Constraints** section of the editor, you can check one or more of the following:

Storage Constraint	Description
Virtual	If set, the attribute will not hold data. Instead, you must define a Read Access method, a Write Access method, or both. See <a href="#">Virtual Attributes</a> on page 195.
Primary Key	If set, the attribute will be used as an identifier to determine if two BPM Objects are the same. You would typically set this for an attribute containing a unique value, such as an ID number.
Not Null	If set, a null value is not allowed, and you will need to set a default value in the <b>Default Value</b> section.

6. In the **Default Value** section, you can set an initial value the attribute will be given when the BPM Object is created. If you did not set the **Not Null** option, you may also set the default value to Null.

7. In the **Required Expression** field, you can enter a Process Business Language expression that will be evaluated when the attribute field is changed in a presentation.

For example, you can make sure the user enters a value greater than zero with the following expression:

```
attributeValue > 0
```

8. Save (📁) the changes and close the editor window by clicking on the **X** in the *bottom* tab. If you close from the top tab, you will close the BPM Object.

### Valid Values

BPM object attributes can be configured with a set of predefined valid values. When you do this, the attribute will be presented to the user as a drop-down list, rather than a field. Thus, the user will only be able to select from the choices provided.

There are three possible **Valid Values** settings for a BPM object attribute:

Valid Values Setting	Description	Presented As
All	Any value within the bounds of the data type is accepted. This is the default setting.	Text field
Static List	One of a list of values is accepted. The list is fixed at design time.	Drop-down list
Dynamic Method	One of a list of values is accepted. The list is dynamically built by a method in run-time.	Drop-down list

The dynamic method is more flexible. For example, you can pull the information from a database. The static list is easy to configure without writing any code.

### Value Descriptions

You will often want end-users to choose among a set of values they can easily recognize, while in the background you need an identifier to return to your system. A typical situation is with a database, where

you might want to obtain a record ID that has no meaning to the user. In other cases, you may need an expense account number or an abbreviated code of some kind, such as a country code, while you want the user to be able to choose from a more descriptive list, such as actual country names.

This can be easily done either with the static list or dynamic method options by choosing to use *value descriptions*.


### Setting a Static Valid Values List

A static valid values list is set at design time. You can use a static valid values list when you know that the list is unlikely to change very often, and when it will contain relatively few options.

The main advantage of a static valid values list is easy to configure, and does not require writing any code. If you want the list of valid values to change frequently, you are better off using a dynamic list obtained from a database or other data source.

To define a list of valid values:

1. In the **Valid Values** section of the attribute editor, select **Static List**. and check the **Edit Value Descriptions** option.  
The Values table will appear.
2. You can choose to add value descriptions by checking the **Edit Value Descriptions** checkbox.  
If you set this option, a *Description* column will appear in the table.
3. To add an entry to the table, click on the Add icon (+) and enter a numeric value in the *Value* column and, if you've set value descriptions, also add a descriptive text in the *Description* column.  
To delete an entry in the table, select the entry and click on the Remove icon (-).
4. Once you are done adding entries to the list, make sure the **Default Value** field is set to a value (not a description) that exists on the list.

 **Note:** When an attribute is configured with a valid values list, the **Default** field must be set to one of the valid values. If it isn't, an error condition is indicated.

5. Save your changes and close the editor for this attribute.

### Defining a Valid Values Method

When a list of valid values will change frequently, you will usually choose to load it dynamically. To do so, you must write a Process Business Language method which will return the list or choose an existing method or array attribute.


The advantages of a using a Process Business Language method to populate a list of valid values are that you can use data from any source, and that you read this data when it is needed, so it is always current. You can also write a method that generates values with an algorithm and does not obtain data from an external source.

You can load a list of valid values dynamically using:

- A method that obtains the values from an external source, such as a database or a Web service, or that generates the values in some other way.
- An array attribute of the same data type as the attribute the valid values are for. For instance, an `Int [ ]` array can hold valid values for an `Int` attribute.

To define a dynamic method valid values list:

1. In the **Valid Values** section of the attribute editor, select **Dynamic Method**.  
The **Method Descriptions** option and the **Method or attribute used to load valid values** panel appear. In the panel, the drop-down list contains currently available methods which return an array of the data type required, as well as array attributes also of the required data type.
2. If you want the valid values list to contain value-description pairs, click on the **Method Descriptions** option.  
If you check this option, an associative array is expected. If not, an indexed array is expected. Therefore, the list of matching methods and attributes will depend on your choice.

3. You can select an existing method or attribute from the drop-down list. If you need to create a new method, go to step 4. Otherwise, select the method or attribute and save your changes.
  4. To create a new method click **New**.  
The **Method** dialog box appears.
  5. Enter the name of the new method in the **Method Name** field, and click **OK**.  
A Process Business Language method editor opens. In the **Properties** window, note that Studio set the return type to match that required by the attribute.
  6. If your method will access a database or other catalog component with external dependencies, set *Server Side Method* to *Yes* in the **Properties** window.
-  **Note:** A server side Process Business Language method executes in the process execution engine. If a method is not server side, it executes where it is called, and may execute in the Workspace server. For the purposes of this setting, the Workspace considered to be the "client". Process Business Language methods are never executed at the browser.
7. Enter your method into the editor. Your method must contain a return statement at the end which specifies the variable to be returned. See the examples below.
  8. Save your changes and close the Process Business Language editor and the attribute editor.

### Examples

The following Process Business Language method loads a list from SQL table `suppliers` into the valid values list of an integer attribute. This table was previously introspected into the project catalog as an SQL component. In this table, `supplierId` is an `Int`, and `supplierName` is a `String`.

In this case the valid values list has descriptions, so an associative array in the form `String[ Int ]` must be returned:

```
listValues as String[Int]

for each record in
  SELECT supplierId, supplierName
  FROM suppliers
do
  listValues[record.supplierId] = record.supplierName
end

return listValues
```

The following loads the first 11 numbers of the Fibonacci sequence into a valid values list with no description, so it returns an indexed array:

```
fNumber as Int[]

fNumber[0] = 1
fNumber[1] = 1

for n in 2..10 do
  fNumber[n] = fNumber[n - 1] + fNumber[n - 2]
end

return fNumber
```

### Virtual Attributes

When an attribute is defined as virtual, it means that it stores no data value, and is defined by its read and write access methods.

BPM Object attributes can be *real* or *virtual*. Defining an attribute as real means that the attribute contains an actual data value. When an attribute is defined as virtual, it means that it contains no data value, and is defined by its read and write access methods.

### Virtual Attribute Definition

You create a virtual attribute the same way you create a real attribute, except that you select **Virtual** in the **Storage Constraints** section of the attribute editor. When you do this, Studio creates two methods

For example, if you designed a BPM Object to obtain weather data, you would likely include a temperature attribute. You could implement the temperature in degrees Celsius in an attribute called `tempCelsius`.

If you wanted to also handle the temperature in degrees Fahrenheit, you could implement a virtual attribute `tempFahrenheit` which would be implemented as follows:

Read access:

```
return tempCelsius * 1.8 + 32
```

Write access:

```
tempCelsius = (value - 32) / 1.8
```

### Virtual Attributes from Process Business Language Code

In Process Business Language code, you access a virtual attribute the same way you access a real attribute. However, you can only access the variable in whatever modes are defined for it (read access or write access or both). For example, a `total` virtual attribute in the `invoice` BPM Object has the following read access method:

```
amount as Decimal(2)
amount = 0

for each item in items do
  amount = amount + item.amount
end

return amount
```

In this fragment of code, `items` is a group, and `amount` is a decimal value representing the cost of each item.

To obtain total value from code, you use:

```
invoice.total
```

There is no write method for the `total` attribute, nor would it make sense to have one (since by definition the total must equal the sum of the amounts), so you would not set it to any value in code.

The `tempFahrenheit` virtual attribute we defined above does have both read access and write access. To write, use:

```
weather.tempFahrenheit = 70.7
```

Remember that the `70.7` value will not be stored. Instead, when this line of code is executed, `weather.tempCelsius` will take on a value of `21.5`.

To read, use:

```
display weather.tempFahrenheit
```

The `70.7` value will be displayed.

## Attribute Data Types

Available data types to define an attribute are:

Type	Description
Bool	A boolean (True or False) value
Int	Integer number
Decimal	Decimal number value with defined precision
Real	Real (floating point) number
String	Text string
Time	A date, time, or date-time value
Interval	A time interval
Binary	Binary container, for example, an image, a file, etc.
Any	Can hold any data type. Similar to a Visual Basic variant.

When defining the attribute type by clicking the browse button next to the Type field, you can select any cataloged component in the Project Catalog as the defined type for the attribute you are declaring. This gives you the ability to typify an attribute within a BPM Object with any component defined in the Project Catalog, even another BPM Object. If the attribute is typed after a different BPM Object, it is referred to as an Inner BPM Object. Find an example in the Inner BPM Object section below.

### Required Fields When Defining an Attribute

The following table shows the required fields for defining an attribute:

Field	Description	Attribute type
Type	Domain type of data that the attribute contains.	Real & Virtual attributes.
Not null	The attribute may be assigned, or not, a null value.	Attribute must have write access.
Primary Key	Is the attribute part of the primary key of the BPM Object?	Attribute must have read access.
Time Precision	Only for Time attributes, you can choose the following: <b>Timestamp</b> : The entire date and time, <b>Date only</b> : The date, <b>Time only</b> : The time.	Date attributes.
Absolute	The value of a time attribute which is set as absolute is stored in GMT-0.	
Maximum length	For String. The <b>Maximum length</b> field defines the maximum <b>number</b> of characters allowed for the attribute.	Real & Virtual attributes.
Default Value	For Int, Decimal, Real, String, Time and Interval types. Initial value that the attribute contains. A default value for a Bool type attribute is defined by selecting the <b>True</b> or <b>False</b> radio button.	Attribute must have write access.
Require Expression	The <b>Require Expression</b> field allows you to enter a Boolean expression to be checked as a precondition to the assignment of a value to the attribute. It can be built either by a simple, one line expression that	Attribute must have write access.

Field	Description	Attribute type
Check Expression	evaluates an expression or by calling a method. This method must return a boolean type.  <b>Check Expression</b> defines integrity validation of the attribute within the BPM Object instance. It can be built either by a simple, one line expression that evaluates an expression or by calling a method. This method must return a boolean type.	Any attribute.
Valid Values	List of valid values that an attribute can be assigned. It can be built either by a hard-coded list of valid values that the attribute can contain or by a call to a method that returns a list (array) of valid values. This method must return an array of the same domain type as the attribute's type. The implementation allows you to show the actual value or its description. For further details, see section <b>Valid Values</b> on this page.	Any attribute

## BPM Object Presentations

A presentation is form that allows the users of the process to view or edit that data in a BPM Object.

### Creating a Presentation

The following procedure shows you how to add a presentation to a BPM Object.

To add a presentation to a BPM Object:

1. Right-click on the BPM Object.
2. Select **New ► Presentation**
3. Enter the name of the new presentation in the **Presentation** field.  
The Presentation Wizard appears.
4. Click **Next**
5. Click **Next**.  
The **Presentation Referenced Attributes** page of the appears.
6. Select the attributes that you want to appear in the presentation.  
You can modify the order of the attributes using the **Up** and **Down** buttons.
7. Click **Finish**.  
The new presentation appears.

## External Resources

External Resources provide a common method for connecting to other resources in an enterprise including databases, Web Services, etc. External Resources are used to define connectivity and configuration information.

It is useful to define these separately since connectivity and configuration information is different between systems. This allows you to easily deploy a project into a new environment because you only need to redefine the External Resource without having to edit application code.

## Creating an External Resource

Studio allows you to define external resources to connect to external systems and resources within your environment.

1. In the **Project Navigator View**, expand the project where you want to define a new External Resource.
2. Right-click **External Resources** and select **New External Resource**.  
The **Edit External Resource** window appears.
3. Enter a name for the External Resource.
4. Select the type of External Resource you want to define.
5. Provide values for each of the External Resource properties.  
For complete information on the properties for each type of External Resource see [External Resource Reference](#) on page 199.
6. Click **OK**.

The new External Resource appears in the Project Navigator.

## External Resource Reference

The following sections provide detailed information about the configuration options for each External Resource type.

### SQL Database

Provides detailed information for the SQL Database External Resource.

### General Properties

The following table defines the general properties that are required for each supported database driver.

Property	Description
Name	Defines the name of the external resource.
Type	Specifies the type of external resource.
Supported Types	Specifies the type of JDBC connection which correspondence to the database vendor and version number.

### Oracle DB2 Driver Properties

You can specify the following connectivity properties for your DB2 database:

#### Basic

Property	Description
Host	Specifies the database server host.
Port	Specifies the port of the database host.
User	Defines user ID you want to use to connect to the database. This user must already exist in DB2 and have permissions to create the schema and tables used to store information.

Property	Description
Password	Specifies password for the user.
Database	Specifies the database you wish to connect to.
Schema	Specifies the database schema to use. (optional)
URL	Defines the URL for the database entry.

### Properties

You can define name/value pairs to provide additional configuration properties to your database. See your vendor's documentation for more information.

 **Note:** Connection property names are case-insensitive.

### Runtime

Property	Description
Maximum Pool Size	Determines the maximum number of connections that can be created within the connection pool.
Maximum Connections Per User	Determines the maximum number of connections that can be created per user.
Connection Idle Time (minutes)	Specifies the maximum time, in minutes, that a database connection can remain idle before it is closed automatically.
Minimum Pool Size	Determines the minimum number of connections that can be created within a connection pool.
Maximum Opened Cursors	Determines the maximum number of cursors that can be opened at one time.

### Oracle Informix Driver Properties

You can specify the following connectivity properties for your Informix database:

#### Basic

Property	Description
Host	Specifies either the IP address or the server name (if your network supports named servers) of the primary database server. For example, 122.23.15.12 or InformixServer.
Port	Specifies the TCP port on which the database server listens for connections. The default varies depending on operating system.
User	Specifies the case-insensitive default user name used to connect to your Informix database.
Password	Specifies a case-insensitive password used to connect to your Informix database.
Database	Specifies the name of the database to which you want to connect.



Property	Description
Server	Specifies the name of the Informix database server to which you want to connect.
URL	Defines the URL format used to connect to your database.

### Advanced

Property	Description
Root dbspace name	Specifies the root dbspace of your Informix database. The rootdb space is the initial dbspace created by the Informix server. The root dbspace contains reserve pages and internal tables.

### Properties

You can define name/value pairs to provide additional configuration properties to your database. See your vendor's documentation for more information.



**Note:** Connection property names are case-insensitive.

### Runtime

Property	Description
Maximum Pool Size	Determines the maximum number of connections that can be created within the connection pool.
Maximum Connections Per User	Determines the maximum number of connections that can be created per user.
Connection Idle Time (minutes)	Specifies the maximum time, in minutes, that a database connection can remain idle before it is closed automatically.
Minimum Pool Size	Determines the minimum number of connections that can be created within a connection pool.
Maximum Opened Cursors	Determines the maximum number of cursors that can be opened at one time.

### Oracle SQL Server Driver Properties

You can specify the following connectivity properties for your SQL Server database:

#### Basic

Property	Description
Host	Specifies the hostname or IP address of the database server.
Port	The TCP port of the primary database server that is listening for connections to the Microsoft SQL Server database.  The default is 1433.

Property	Description
User	Specifies the case-insensitive user name used to connect to your Microsoft SQL Server database.
Password	Specifies a case-insensitive password used to connect to your Microsoft SQL Server database.
Database	Specifies either the IP address or the server name, if your network supports named servers, of the primary database server.
URL	Defines the URL format used to connect to your database.

### Properties

You can define name/value pairs to provide additional configuration properties to your database. See your vendor's documentation for more information.



**Note:** Connection property names are case-insensitive.

### Runtime

Property	Description
Maximum Pool Size	Determines the maximum number of connections that can be created within the connection pool.
Maximum Connections Per User	Determines the maximum number of connections that can be created per user.
Connection Idle Time (minutes)	Specifies the maximum time, in minutes, that a database connection can remain idle before it is closed automatically.
Minimum Pool Size	Determines the minimum number of connections that can be created within a connection pool.
Maximum Opened Cursors	Determines the maximum number of cursors that can be opened at one time.

### Oracle Driver Properties

You can specify the following connectivity properties for your Oracle database:

#### Basic

Property	Description
Host	Specifies the hostname or IP address of the database server.
Port	Specifies the TCP port of the Oracle listener running on the Oracle database server. The default is 1521, which is the Oracle default port number when installing the Oracle database software.
User	Specifies the case-insensitive default user name used to connect to your Oracle database.
Password	Specifies the case-insensitive password used to connect to your Oracle database.

Property	Description
SID	Specifies the Oracle System Identifier that refers to the instance of the Oracle database running on the server.
Schema (optional)	Specifies the schema of the Oracle database server.
URL	Defines the URL used to connect to your database.

### Advanced

Property	Description
Tablespace	Specifies the tablespace Oracle BPM uses to create new tables.
Temporary Tablespace	Specifies the temporary tablespace Oracle BPM uses to create new tables.
Profile	Specifies the profile Oracle BPM assigns to the users it creates.
Use TimeStamp for Date Columns	Specifies if a column of type Date stores the date and the time. If unselected it only stores the date.

### Properties

You can define name/value pairs to provide additional configuration properties to your database. See your vendor's documentation for more information.



**Note:** Connection property names are case-insensitive.

### Runtime

Property	Description
Maximum Pool Size	Determines the maximum number of connections that can be created within the connection pool.
Maximum Connections Per User	Determines the maximum number of connections that can be created per user.
Connection Idle Time (minutes)	Specifies the maximum time, in minutes, that a database connection can remain idle before it is closed automatically.
Minimum Pool Size	Determines the minimum number of connections that can be created within a connection pool.
Maximum Opened Cursors	Determines the maximum number of cursors that can be opened at one time.

### Oracle Sybase Driver Properties

You can specify the following connectivity properties for your Sybase database:

**Basic**

Property	Description
Host	Specifies the hostname or IP address of the Sybase database server.
Port	Specifies the TCP port of the Sybase listener running on the Sybase database server. The default is 2048, which is the Sybase default port number when installing the Sybase database software.
User	Specifies the case-sensitive default user name used to connect to your Sybase database.
Password	Specifies the case-sensitive password used to connect to your Sybase database.
Database	The name of the database that contains the tables.
Device	The device where tables are created.
Allocation Size	The amount of space to allocate to the database extension.
URL	The URL to connect to your Sybase database. You cannot edit this field directly, the URL is formed using the host and database you define.

**Properties Tab**

The **Properties** tab allows you to define name/value pairs to configure database properties.



**Note:** All connection property names are case-insensitive.

**Runtime Tab**

Property	Description
Maximum Pool Size	Determines the maximum number of connections that can be created within the connection pool.
Maximum Connections Per User	Determines the maximum number of connections that can be created per user.
Connection Idle Time (minutes)	Specifies the maximum time, in minutes, that a database connection can remain idle before it is closed automatically.
Minimum Pool Size	Determines the minimum number of connections that can be created within a connection pool.
Maximum Opened Cursors	Determines the maximum number of cursors that can be opened at one time.

**Oracle DB2 AS/400 JDBC Properties**

You can configure the following properties for your DB2 AS/400 database:

**Basic**

Property	Description
Host	

Property	Description
Port	
User	Specifies the case-sensitive user name used to connect to your DB2 AS/400 database.
Password	Specifies the case-sensitive password used to connect to your DB2 AS/400 database.
Database	Specifies the name of your DB2 AS/400 database.
Schema	Specifies the schema of the DB2 AS/400 server.
URL	The URL to connect to your DB2 AS/400 database. You cannot edit this field directly, the URL is formed using the host, port and database you define.

### Properties

You can define name/value pairs to provide additional configuration properties to your database. See your vendor's documentation for more information.



**Note:** Connection property names are case-insensitive.

### Runtime

Property	Description
Maximum Pool Size	Determines the maximum number of connections that can be created within the connection pool.
Maximum Connections Per User	Determines the maximum number of connections that can be created per user.
Connection Idle Time (minutes)	Specifies the maximum time, in minutes, that a database connection can remain idle before it is closed automatically.
Minimum Pool Size	Determines the minimum number of connections that can be created within a connection pool.
Maximum Opened Cursors	Determines the maximum number of cursors that can be opened at one time.

### Oracle DB2 OS390 Properties

You can configure the following connectivity properties for your DB2 OS390 database:

#### Basic

Property	Description
User	Specifies the case-sensitive user name used to connect to your DB2 OS390 database.
Password	Specifies the case-sensitive password used to connect to your DB2 OS390 database.
Database	Specifies the name of your DB2 OS390 database.
Schema	Specifies the schema of the DB2 OS390 server.

Property	Description
URL	The URL to connect to DB2 OS390 database. You cannot edit this field directly, the URL is formed using the database name you define.

### Properties

You can define name/value pairs to provide additional configuration properties to your database. See your vendor's documentation for more information.



**Note:** Connection property names are case-insensitive.

### Runtime

Property	Description
Maximum Pool Size	Determines the maximum number of connections that can be created within the connection pool.
Maximum Connections Per User	Determines the maximum number of connections that can be created per user.
Connection Idle Time (minutes)	Specifies the maximum time, in minutes, that a database connection can remain idle before it is closed automatically.
Minimum Pool Size	Determines the minimum number of connections that can be created within a connection pool.
Maximum Opened Cursors	Determines the maximum number of cursors that can be opened at one time.

### Derby Database Driver Properties

You can specify the following properties for your Derby database:

#### Basic

Property	Description
Database Path	The path to the file that contains the database.
Requires Authentication	Specifies if you need to provide a user and password to connect to the database.
User	Specifies the case-sensitive user name used to connect to your Derby database.
Password	Specifies the case-sensitive password used to connect to your Derby database.
Schema	Specifies the schema of the derby database server.
URL	The URL to connect to the database. You cannot edit this field directly, the URL is formed using the database path you define.

### Properties

You can define name/value pairs to provide additional configuration properties to your database. See your vendor's documentation for more information.

 **Note:** Connection property names are case-insensitive.

## Runtime

Property	Description
Maximum Pool Size	Determines the maximum number of connections that can be created within the connection pool.
Maximum Connections Per User	Determines the maximum number of connections that can be created per user.
Connection Idle Time (minutes)	Specifies the maximum time, in minutes, that a database connection can remain idle before it is closed automatically.
Minimum Pool Size	Determines the minimum number of connections that can be created within a connection pool.
Maximum Opened Cursors	Determines the maximum number of cursors that can be opened at one time.

## Generic JDBC Version 1 Properties

You can specify the following connectivity properties for your generic JDBC driver:

### Basic

Property	Description
JDBC Driver	Specifies the class name for the JDBC driver.
URL	Specifies the URL used to connect to your database.
User	Specifies the case-insensitive default user name used to connect to your database.
Password	Specifies the case-insensitive password used to connect to your database.

## Runtime

Property	Description
Maximum Pool Size	Determines the maximum number of connections that can be created within the connection pool.
Maximum Connections Per User	Determines the maximum number of connections that can be created per user.
Connection Idle Time (minutes)	Specifies the maximum time, in minutes, that a database connection can remain idle before it is closed automatically.
Minimum Pool Size	Determines the minimum number of connections that can be created within a connection pool.
Maximum Opened Cursors	Determines the maximum number of cursors that can be opened at one time.

## Remote JDBC Properties

You can specify the following connectivity properties for a remote JDBC connection:

Property	Description
Database Type	Specifies the type of database used by the data source defined in the J2EE Application Server.
J2EE	Specifies the external resource defined for the J2EE Application Server.
Lookup Name	Specifies the JNDI name used to locate the data source in the J2EE Application Server.

### SAP Service

You can specify the following connectivity properties for an SAP Service.

#### General Properties

This section defines the general properties for this External Resources:

Property	Description
Name	Defines the name of this external resource.
Type	Specifies the external resource type.
Supported Types	This field is disabled for this external resource type.

### SAP Properties

The following properties must be defined for an SAP Service:

Property	Description
Client	Specifies the SAP log-on client.
User Id	Specifies the SAP log-on user.
Password	Specifies the SAP log-on password.
Hostname	Specifies the host name of the application server.
System Number	Specifies the SAP log-on system number.
Language	Specifies the SAP log-on language.
Pool Size	Specifies the size of the connection pool the SAP client uses.

### Web Service

Provides detailed information for configuring a Web Service.

#### General Properties

This section defines the general Web Service properties:

Property	Description
Name	Defines the name of the external resource.
Type	Specifies the type of external resource.
Supported Types	Specifies the type of Web Service.



### Producer Web Service Properties

The following properties must be configured for a Producer Web Service External Resource:

Property	Description
Authentication Type	Defines the authentication type of this web service. <ul style="list-style-type: none"> <li>• None - uses no authentication for the web service</li> <li>• Username Token Profile</li> </ul>
Endpoint Identifier	The name used to expose the web service endpoint.

### Consumer Web Service Properties

The following properties must be configured for a Consumer Web Service under the **Endpoint** tab:

Property	Description
Static Endpoint Binding	Specifies if the endpoint and transport are static.
UDDI Dynamic Endpoint Binding	Specifies that the endpoint is obtained by searching the binding in the UDDI registry.

The following properties must be configured for a Consumer Web Service that uses Static Endpoint Binding:

Property	Description
Transport Type	Specifies the type of transport used to communicate with the web service. Possible values are: <ul style="list-style-type: none"> <li>• HTTP</li> <li>• JMS</li> <li>• ALSB</li> </ul>

#### HTTP Transport Configuration

Property	Description
HTTP Server Configuration	Specifies the Server Configuration external resource where the web service is running.
Path	Specifies the path of the web service.

#### JMS Transport Configuration

Property	Description
JNDI Server Configuration	Specifies the JNDI Directory Service external resource that contains the configuration used to lookup the JMS resource.
Destination Type	The method used to subscribe to the queue. Possible values are: <ul style="list-style-type: none"> <li>• Queue</li> <li>• Topic</li> </ul>
JMS Factory Name	Specifies the name of the JMS Factory used to obtain the JMS resource.

Property	Description
Destination Name	Specifies the name of the JMS resource.

## ALBS Transport Configuration

Property	Description
Proxy Server Configuration	Specifies the external resource of type Aqualogic Service Bus and subtype Proxy Server that corresponds to your Aqualogic Service Bus Server.
Service Name	Specifies the name of the proxy server used for transport.
Propagate Transaction	Specifies if transactions can be propagated from the PBL component to Aqualogic Service Bus.

The following properties must be configured for a Consumer Web Service that uses UDDI Dynamic Endpoint Binding:

Property	Description
Server Configuration	Specifies the Server Configuration external resource where the web service is running.
Inquiry Path	Specifies the location of the UDDI inquiry.
Requires Authentication	Specifies if the inquiry requires authentication.
Security Path	Specifies the path where the UDDI authentication server is running.
User	Specifies the case-sensitive user name used to connect to your web service.
Password	Specifies the case-sensitive password used to connect to your Oracle database.
Service Key	Specifies the key of the UDDI service.
Binding TModel Key	Specifies the key of the UDDI binding.

The following properties must be configured for a Consumer Web Service under the **Runtime** tab:

Property	Description
Use System Exceptions	Specifies if the Oracle BPM Engine treats SOAP faults as System Exceptions. If this checkbox is not selected the Oracle BPM Engine treats SOAP faults as Business Exceptions.

The following properties must be configured for a Consumer Web Service under the **Security** tab:

Property	Description
Send Username Token	<p>Specifies if the web service uses Username Token Profile web service security. Possible values are:</p> <ul style="list-style-type: none"> <li>• None - uses no authentication for the web service</li> <li>• Plain - uses Plain Username Token authentication for the web service</li> </ul>

Property	Description
	<ul style="list-style-type: none"> <li>Digest - uses Digest Username Token authentication for the web service</li> </ul>
Send Nonce and Timestamp	Specifies if the web service security headers include the nonce and timestamp elements.
Username	Defines the case-sensitive user name required to connect to your web service.
Password	Defines the case-sensitive password required to connect to your web service.
Confirm Password	Defines the case-sensitive password required to connect to your web service.

## Server Configuration

Provides detailed information for configuring an Server as an External Resource.

### General Properties

Property	Description
Name	Defines the name of this external resource.
Type	Specifies the external resource type.

### Details

The following properties must be defined for an Server configuration:

Property	Description
Protocol	<p>Specifies the protocol for the server configuration. Possible values are:</p> <ul style="list-style-type: none"> <li>HTTP</li> <li>HTTPS</li> </ul>
Host	Specifies the hostname or IP address of the server.
Port	Specifies the port number where the server listens for requests. The default is 85.
Path	Specifies the path part of the URL used to locate the Server.
URL	Specifies the URL of the server. You cannot edit this field directly, the URL is formed based on the protocol, host, port and path you define.
Requires HTTP Basic Authentication	Specifies whether the HTTP server requires basic HTTP authentication.
User	Specifies the username used to authenticate HTTP requests.
Password	Specifies the password used to authenticate HTTP requests.

Property	Description
Connection Timeout	Specifies the time (in seconds) the server waits for a connection to become available.

### Microsoft .NET Service

Provides detailed information for configuring a Microsoft .NET Service.

#### General Properties

This section defines the general properties for this External Resources:

Property	Description
Name	Defines the name of this external resource.
Type	This field is disabled for this external resource type.

### Microsoft .NET Service Properties

The following properties must be defined for a Microsoft .NET Service:

Property	Description
Host	Defines the location of the .NET Bridge host.
Port	Defines the port used by the .NET Bridge host.

### Mail Outgoing Service

Provides detailed information for configuring a Mail Outgoing Service.

#### General Properties

This section defines the general properties for this External Resource:

Property	Description
Name	Defines the name of this external resource.
Type	Specifies the external resource type.
Supported Types	Specifies the mail protocol used by the outgoing mail server. Possible values are: <ul style="list-style-type: none"> <li>SMTP</li> </ul>

### SMTP Properties

The following properties must be for an SMTP Mail Outgoing Service:

Property	Description
Server Host	Specifies the hostname or IP address of the outgoing mail server.
Server Port (optional)	Specifies the port number the outgoing mail server uses to send new mails. The default values is 25.
User	Specifies the case-sensitive user name used to connect to your mail server.

Property	Description
Password	Specifies the case-sensitive password used to connect to your mail server.
Secure Connection	Defines the security protocol used. Valid values are: <ul style="list-style-type: none"> <li>• No - No security protocol is used</li> <li>• TLS, if available</li> <li>• TLS</li> <li>• SSL - Uses the Secure Sockets Layer (default)</li> </ul>

### J2EE Application Server

Provides detailed information on configuring a J2EE Application Server as an External Resource.

To create Enterprise JavaBeans components, you must define a J2EE Application Server as an External Resource.

### General Properties

The following table defines the general properties for this External Resource:

Property	Description
Name	Defines the name of the external resource.
Type	Specifies the type of external resource.
Supported Types	Specifies the type of J2EE server. Possible values are: <ul style="list-style-type: none"> <li>• Generic J2EE</li> <li>• Local J2EE</li> </ul>

### Local J2EE Application Server Properties

This type of J2EE application server is used when the process is deployed in a J2EE environment, and the resources are located in the same Application Server.

The following properties must be configured for a Local J2EE Application Server using the **Basic** tab:

Property	Description
User Transaction Lookup Name	The JNDI lookup name used to locate the user transaction object used to demarcate transactions in the Application Server.

### Generic J2EE Application Server Properties

The following properties must be configured for a J2EE Application Server using the **Basic** tab:

Property	Description
Initial Context Factory	Specifies the classname of the JNDI Factory used to connect to the J2EE Application Server.
URL	Specifies the URL to connect to the J2EE Application Server.
Principal	Specifies the case-sensitive user name used to connect to your J2EE Application Server.

Property	Description
Credentials	Specifies the case-sensitive password used to connect to your J2EE Application Server.

The following properties must be configured for a J2EE Application Server under the **Advanced** tab:

Property	Description
User Transaction Lookup Name	The JNDI lookup name used to locate the user transaction object used to demarcate transactions in the Application Server.

### Properties Tab

The **Properties** tab allows you to define name/value pairs to configure J2EE Application Server connection properties.



**Note:** All connection property names are case-insensitive.

### Runtime Tab

The following properties must be configured for a J2EE Application Server using the **Runtime** tab:

Property	Description
Maximum Pool Size	Determines the maximum number of connections that can be created within the connection pool.
Maximum Connections Per User	Determines the maximum number of connections that can be created per user.
Connection Idle Time (minutes)	Specifies the maximum time, in minutes, that a connection can remain idle before it is closed automatically.
Minimum Pool Size	Determines the minimum number of connections that can be created within a connection pool.

### Enterprise JavaBean (EJB)

Provides detailed information for configuring an Enterprise JavaBean as an External Resource.

#### General Properties

This section defines the general properties for this External Resource:

Property	Description
Name	Defines the name of this external resource.
Type	Specifies the external resource type.
Supported Types	This field is disabled for this external resource type.

#### EJB Properties

The following properties must be defined for an EJB:

Property	Description
J2EE	Specifies the external resource defined for the J2EE Application Server.
Lookup Name	Specifies the JNDI name used to locate the Enterprise JavaBean in the J2EE Application Server.

### Java Class Library

Provides detailed information for configuring a Java Class Library as an External Resource.

#### General Properties

This section defines the general properties for this External Resource:

Property	Description
Name	Defines the name of this external resource.
Type	Specifies the external resource type.
Supported Types	This field is disabled for this external resource type.

### Java Class Library Properties

The following properties must be defined for a Java Class Library:

Property	Description
Versionable	Specifies if the jars contained in this External Resource are published using project versioning. Refer to <a href="#">Versionable Java Libraries</a> on page 169 for details.
JAR Libraries	Specifies the JAR files that contain your Java library and the JAR files from the libraries it depends on.

### AquaLogic Service Bus

Provides detailed information for configuring an AquaLogic Service Bus as an External Resource.

#### General Properties

This section defines the general External Resource properties:

Property	Description
Name	Defines the name of the external resource.
Type	Specifies the type of external resource.
Supported Types	Specifies the type of AquaLogic Service Bus connection.

### Management Host Properties

The following properties must be configured for a Management Host:

Property	Description
Host	
Port	

Property	Description
User	
Password	

### Proxy Service Properties

The following properties must be configured for a Proxy Service:

Property	Description
Host	
Port	
User	
Password	

### Process Deployment Properties

The following properties must be configured for a Process Deployment:

Property	Description
Management Configuration	
Project Name	
WSDL Folder	
Business Services Folder	
WS-Security Account	
Transport	
Host	
Port	

### Mail Incoming Service

Provides detailed information for configuring a Mail Incoming Service as an External Resource.

### General Properties

This section defines the general properties for this External Resource:

Property	Description
Name	Defines the name of this external resource.
Type	Specifies the external resource type.
Supported Types	Specifies the mail protocol used by the incoming mail server. Possible values are: <ul style="list-style-type: none"> <li>• IMAP</li> <li>• POP3</li> </ul>

### IMAP and POP3 Properties

The following properties must be when using IMAP or POP3:



Property	Description
Service Host	Specifies the hostname or IP address of the incoming mail server.
Server Port (optional)	Specifies the port number where the incoming mail server listens to receive new mails. The default values are: <ul style="list-style-type: none"> <li>• IMAP: 143</li> <li>• POP3: 110</li> </ul>
User	Specifies the case-sensitive user name used to connect to your mail server.
Password	Specifies the case-sensitive password used to connect to your mail server.
Secure Connection	Specifies if this server needs a secure connection. Possible values are: <ul style="list-style-type: none"> <li>• SSL</li> <li>• No - No security protocol is used</li> </ul>
Secure Authentication	Specifies if your incoming mail server needs authentication.

### Microsoft COM Service

Provides detailed information for configuring a Microsoft COM Service.

### General Properties

This section defines the general properties for this External Resources:

Property	Description
Name	Defines the name of this external resource.
Type	This field is disabled for this external resource type.

### Microsoft COM Service Properties

The following properties must be defined for a Microsoft COM Service:

Property	Description
Host	Defines the location of the COM Bridge host.
Port	Defines the port used by the COM Bridge host.

### JMX Service

Provides detailed information on configuring a JMX Service as an External Resource.

Oracle BPM supports the following JMX service providers:

- Oracle Weblogic
- IBM WebSphere
- JBoss
- JSR-160

- MX4J

### General Properties

This section defines the general properties for this External Resource:

Property	Description
Name	Defines the name of the external resource.
Type	Specifies the type of external resource.
Supported Types	Specifies the type of JMX Server. Possible values are: <ul style="list-style-type: none"> <li>• Oracle Weblogic</li> <li>• IBM WebSphere</li> <li>• JBoss</li> <li>• JSR-160</li> <li>• MX4J</li> </ul>

### Oracle Weblogic, IBM WebSphere, JBoss and MX4J Basic Properties

The following properties must be configured for an Oracle Weblogic, IBM WebSphere, JBoss or MX4J JMX Service using the **Basic** tab:

Property	Description
Host	Specifies the hostname or IP address of the MBean Server.

### JSR-160 Basic Properties

The following properties must be configured for a JSR-160 JMX Service using the **Basic** tab:

Property	Description
Service URL	Specifies the URL used to connect to the MBean Server.
Principal	Specifies the case-sensitive user name used to connect to your M Server.
Credentials	Specifies the case-sensitive password used to connect to your MBean Server.

### Advanced

The **Advanced** tab allows you to define name/value pairs to configure J2EE Application Server connection properties.



**Note:** All connection property names are case-insensitive.

### CORBA Service

Provides information on configuring a CORBA Service as an External Resource.

Studio allows you to catalog CORBA objects that reside in an Interface Repository. Once cataloged, you can manipulate the components of the CORBA object in your Method tasks in a process design. To catalog a CORBA object, you need a configuration of CORBA type. Note that creating a configuration allows you to reuse it each time you need to add new components or to introspect existing ones.

## General Properties

This section defines the general SQL Database properties:

Property	Description
Name	Defines the name of the external resource.
Type	Specifies the type of external resource.
Supported Types	This field is disabled for this external resource type.

## CORBA Service Properties

The following properties must be configured for a CORBA Service under the **Basic** tab:

Property	Description
Use Application Server ORB	Specifies if Oracle BPM should lookup the default ORB when running on an application server. If unchecked, the default ORB is used.

The following properties must be configured for a CORBA Service under the **Interface Repository** tab:

Property	Description
Read IOR from URL	Specifies the URL to locate the IOR file used to locate the Interface Repository.
Use This IOR	Specifies the absolute path to locate the IOR file used to locate the Interface Repository.

## JMS Messaging Service

Provides detailed information for configuring a JMS Messaging Service as an External Resource.

### General Properties

This section defines the general properties for this External Resources:

Property	Description
Name	Defines the name of this external resource.
Type	Specifies the external resource type.
Supported Types	This field is disabled for this external resource type.

### JMS Messaging Service Properties

The following properties must be configured for a JMS Messaging Service:

Property	Description
J2EE	Specifies the J2EE Application Server external resource that corresponds to the Application Server where the JMS resource is located.
Destination Type	Specifies the method used to subscribe to the queue. Possible values are: <ul style="list-style-type: none"> <li>Queue</li> <li>Topic</li> </ul>

Property	Description
Lookup Name	Specifies the lookup name to locate the JNDI resource.
Connection Factory Lookup	Specifies the name of the JMS Factory used to obtain the JMS resource.
User	Specifies the case-sensitive user name used to connect to your JMS Server.
Password	Specifies the case-sensitive password used to connect to your JMS Server.
JMS Listener Port (WebSphere only)s	The WebSphere Listener port for the JMS resource.

### JNDI Directory Server

Provides detailed information on configuring a JNDI Directory Server as an External Resource.

### General Properties

This section defines the general properties for this External Resource:

Property	Description
Name	Defines the name of the external resource.
Type	Specifies the type of external resource.
Supported Types	Specifies the type of JNDI Directory Server. Possible values are: <ul style="list-style-type: none"> <li>• Generic JNDI</li> <li>• Active Directory</li> <li>• Sun One LDAP</li> </ul>

### Basic

The following properties must be configured for a JNDI Directory Server using the **Basic** tab:

Property	Description
Initial Context Factory	Defines the name of the initial context factory you want to use.
URL	Defines the URL you want to use to connect to the directory service.
Principle	Defines the root distinguished name for the directory service.
Credentials	Specifies the password for the directory service.
Referrals	<ul style="list-style-type: none"> <li>• follow: the entry will be looked for directly.</li> <li>• ignore: the entry is not looked for.</li> <li>• throw: you must catch and manage any exceptions.</li> </ul>

The following properties must be configured for a JNDI Directory Server using the **Properties** tab:

Property	Description
Properties	Define any name/value pair properties that need to be passed to the directory service.

The following properties must be configured for a JNDI Directory Server using the **Runtime** tab:

Property	Description
Maximum Pool Size	Determines the maximum number of connections that can be created within the connection pool.
Maximum Connections Per User	Determines the maximum number of connections that can be created per user.
Connection Idle Time (minutes)	Specifies the maximum time, in minutes, that a connection can remain idle before it is closed automatically.
Minimum Pool Size	Determines the minimum number of connections that can be created within a connection pool.

### Java Process Definition (JPD)

Provides detailed information for configuring a Java Process Definition as an External Resource.

#### General Properties

This section defines the general properties for this External Resources:

Property	Description
Name	Defines the name of this external resource.
Type	Specifies the external resource type.
Supported Types	This field is disabled for this external resource type.

#### Java Process Definition Properties

The following properties must be configured for a Java Process Definition:

Property	Description
HTTP Server Configuration	Specifies an external resource of type HTTP Server Configuration that corresponds to the container server for the JPD Service.
Path	Specifies the path to the JPD Service in the container server.

## Auditing

Oracle BPM Processes provide auditing capabilities by recording information about the occurrence of sensible events during the execution of the process.

Audit Events allow you to keep track of the events that occur while a process instance is flowing through the process. An event is registered each time the instance enters or exits an activity (simple activity, group, process). The Process Execution Engine generates one event per action each time an even enters or exits an activity. By default events are only generated for interactive activities.

## Enabling Auditing

Auditing properties are set in two locations within Oracle BPM:

- As a property of an activity within a business process.
- Globally for all published projects using Process Administrator.

## When Audit Events Are Generated

You can define which process activities will generate auditing events.

You set whether an activity generates events in design time, and you can set this for each activity, for activity groups, or for the whole process. You can also set whether the process engine generates events or not at run time.

A timestamp is generated for each event. The application server retrieves each timestamp from the operating system.

### Design Time

Design time event generation options are set in Studio. At design time, the following options are available for each *activity*:

Setting	Description
<i>Default</i> (default setting)	Indicates that the activity will record events according to the group or process default, as described below.
<i>Generate Events</i>	The activity will generate events, regardless of the group or process default.
<i>Do not Generate Events</i>	The activity will not generate events, regardless of the process default.

Also at design time, the following options are available for each *group*:

Setting	Description
<i>Default</i> (default setting)	Indicates that the group's activities will record events according to the process setting, as described below.
<i>Generate Events</i>	The default for the group's activities will be to generate events, regardless of the process default.
<i>Do not Generate Events</i>	The default for the group's activities will be not to generate events, regardless of the process default.

The following options are available for each *process*. These settings will be used by activities or groups set to *Default*. If a group or activity is not set to *Default*, it will ignore the process setting.

Setting	Description
<i>Generate Events for Interactive Activities</i> (default setting)	Each activity set to the <i>Default</i> option will generate events if it is interactive, and will not generate events if it is automatic.
<i>Generate Events for all Activities</i>	Each activity or group set to <i>Default</i> will generate events.
<i>Do not Generate Events</i>	Each activity or group set to <i>Default</i> will not generate events.

### Run Time

You set run time event generation in Process Administrator, for each process engine. You can set each process engine to one of the following event recording modes:

Setting	Description
<i>Depends on Process</i> (default setting)	Indicates that the process engine will follow the settings of each process. That is, it will follow the design time settings as described in the section above.
<i>Never</i>	No events are recorded, except instance begin and end activities. Design time settings are ignored.
<i>Always</i>	All activities generate events, regardless of process, group, or activity settings. Design time settings are ignored.

### Remarks

If all settings are left at their defaults, a process will generate events for interactive activities and not for automatic activities. Begin and End activities are always generated.

This is a reasonable default because activities that require human interaction have the most variable execution times. However, you may want to measure some automatic activities, for example those that invoke external systems that for whatever reason have variable execution times.

Each event generated has a slight performance cost. This cost is not important for interactive activities since these activities spend most of their time waiting for participants to execute them. However, it may have significant impact on automatic activities that are performed frequently.

## Which Audit Events are Generated

The following auditing events are generated:

- All the activities generate the same events (IN, OUT, EXECUTE, SELECT, UNSELECT, among others.)
- The Begin activity has no Activity IN event as the instance is created in it.
- The End activity has no Activity OUT event as the instance terminates there.
- The Join activity generates events only if the Split associated activity generates events.
- When an instance is created, a CREATION event is generated instead of an Enter event. This event is always automatically generated if the Engine stores events. All original instances (copy 0) have the CREATION event.
- When an instance is finished, an END event is generated. This event is always automatically generated if the Engine stores events. All terminated original instances (copy 0) have the END event.
- Interactive activities have additional events that occur between the Enter and End events.

If you have any **Generates events** check box selected, the Audit Trail in WorkSpace is enabled. The Audit Trail displays all events that have occurred for an instance.

## Configuring Auditing for a Process

Auditing events generation can be configured at a process level. This configuration defines the default auditing behavior for all the activities in the process.

To configure auditing events for a process:

1. Right-click on the process.
2. Select **Properties**.
3. Choose one of the following options:

Option	Description
<b>Generate Events for Interactive Activities</b>	Enables auditing events generation only for interactive activities.

Option	Description
Generate Events for all Activities	Enables auditing events generation for all the activities in the process.
Do not generate events	Disables auditing events for this Process.

### Configuring Auditing Events for an Activity

Auditing events generation can be enabled independently for each activity.

To configure auditing events for an activity:

1. Right-click on the activity.
2. Select **Properties**.
3. Select **Advanced**
4. Choose one of the following options:

Option	Description
Default	Uses the configuration of the process to which the activity belongs to.
Generate Events	Enables auditing events generation for this activity.
Do not generate events	Disables auditing events for this activity.

### Configuring the Generation of Audit Records for an Activity Group

An activity group is a compound activity. It is a set of activities that may include other activity groups. During the design time, you can configure audit record generation for an activity group when you create the group.

To configure audit record generation for an activity group, use Oracle BPM Studio as follows:

1. In the navigator pane, expand the project and the process in which the activity you want to modify resides.
2. After you decide the activities you want to group:
  - a) Click and hold the mouse button on the background of the process editor.
  - b) Drag the mouse to select the activities you have decided to group. The activities are surrounded by a dotted line.
  - c) Right-click inside the dotted line. A pop-up menu appears.
  - d) Select **Create Group with Selection**. The Activity dialog box appears.
3. In the Activity dialog box, in the Category pane, select **Advanced**. The Advanced pane appears on the right of the dialog box.
4. In the Generate Events section, specify whether and at what level to generate events for the activity. You have these options:
  - Default--Enables audit record generation at the activity level depending on the group to which the activity belongs, if any, or the process event generation definition
  - Generate Events--Enables audit record generation for all events at the activity level only, and not at the process level or within an engine. If no audit records are generated at the group or process level, they will be generated at the activity level.
  - Do not Generate Events
5. Once you have made your selection, click **OK**.



## Modifying the Generation of Audit Records for an Activity Group

To modify the configuration for generating audit records for an activity group, use Oracle BPM Studio and follow these steps:

1. In the navigator pane, expand the project and the process in which the activity group you want to modify resides.
2. Right-click inside the dotted line of the activity group.  
A pop-up menu appears.
3. From the pop-up menu, select **Properties**.  
The Activity dialog box appears.
4. In the Activity window, in the Category pane, select **Advanced**.  
The Advanced pane appears on the right of the window.
5. In the Generate Events section, specify whether and at what level to generate events for the activity.  
You have these options:
  - **Default**--Indicates that the group's activities will record events according to the default process setting.
  - **Generate Events**--The default for the group's activities will be to generate events, regardless of the process default.
  - **Do not Generate Events**--The default for the group's activities will be not to generate events, regardless of the process default.
6. After you have made the changes, click **OK**.

## Advanced Use Cases

---

### Dynamic Business Rules

In Oracle BPM you can define dynamic business rules. A dynamic business rule is a set of one or more conditions evaluated against project variables or business parameters.

You define dynamic business rules in the business rule editor, which has two modes: A simple GUI mode and an advanced mode where you define the rule by writing PBL code. You define each business rule with a unique name.

Dynamic business rules are defined for the project. Once you define a dynamic business rule, you can use it in any process from a [Business Rule Transitions](#) on page 114, or from code in any PBL method. You can design your project so that WorkSpace participants can edit business rules at run time.

#### Business Rule Transitions

In Oracle BPM, conditional transitions can be used to control the flow of an instance within a process. However, standard conditional transitions are defined at design time and cannot be edited in run time.

Business rules can also be used to control process flow when used by business rule transitions. Business rule transitions are similar to conditional transitions except that they evaluate a business rule instead of a conditional expression.

#### Access From PBL Code

The Rules Editor and rule evaluation use the standard component `Fuego.Rules.Rule`. For advanced use cases this component may be used directly from PBL code. This allows you to add or evaluate rules from any type of activity or BPM Object.

#### Business Rules at Run Time

Dynamic business rules are considered dynamic because they can be modified at run time. Dynamic business rules can be edited by any participant who is a member of a role that has been enabled to edit business rules.

### When to use Dynamic Business Rules

Dynamic business rules are suitable for specific situations.

You can use dynamic business rules when you want to do any of the following:

- You want at least some participants to be able to change a business rule at run-time, from the WorkSpace, within a deployed process.
- You want to define a simple business rule without using code, or you want to allow Business Analysts to do so. Dynamic business rules can be edited with a simple Business Rules Editor, or, for advanced capability, by writing code.
- You want to share a single business rule across processes or activities in a project. Dynamic business rules are *named*, and they are defined at the project level.
- You want to audit the use of a business rule. Every time a business rule is evaluated, this evaluation (and its result) is recorded in the Audit Trail.

You should not use dynamic business rules if any of the following are true:

- You need the rule to operate on instance variables. Business rules cannot access instance variables. They can only evaluate project variables and business parameters.
- You don't have a specific reason to use dynamic business rules. They require more system resources because they are audited, require project variables, and are evaluated at run time.

## Using Dynamic Business Rules

Business rules often need to change depending on your business context and requirements. The Business Rules Editor allows you to easily view and change the business rules for each of your Oracle BPM Projects.

### Business Rule Editing Modes

The Business Rules Editor has two modes: *simple editor* and *advanced editor*. The simple editor is a UI which lets you define one or more conditions to be checked. You can determine if you want the rule to match all the conditions or any of them, and you must always compare a variable or business parameter to a constant.

The advanced editor allows you to enter PBL code and can therefore be used to implement more complex conditions.

### Editing Rules from the Workspace

A web version of the Business Rules Editor can also be used from the Oracle BPM Workspace web application. To have access to the editor, a participant must have a role where a global activity has been defined with an *Edit Business Rules* implementation type.

This activity will be visible in the **Applications** panel of the Workspace page.

### Versioning of Rules

Because dynamic business rules can be edited at run time, they are versioned. This allows the user to revert to an earlier version if a newly edited one does not work as expected.

### Version Numbers

Every time a business rule is edited, the business rules version is incremented. Business rules are not versioned individually. Rather, there is a single version number for the whole set of rules in the project. Hence, if you have two business rules A and B, and you edit rule B three times and rule A twice, your version numbers may look like those shown below:

Version number	Rule Edited
1	A
2	B
3	B
4	A
5	B

In this way, the rule A will be available in versions 1 and 4, while rule B will be available in versions 2, 3, and 5. Workspace users are able to select any available version of a given rule in the Workspace business rules editor.

### Rule Compatibility Checking

At run time, every time a rule needs to be evaluated, the first (most recent) compatible version of it is fetched from the Directory Database. A compatible version of a business rule is a version which accesses project variables and business parameters that exist in the project and have the type expected by the rule.

If no compatible rule is found, the rule that was originally published with the current project version is used. In other words, none of the versions of the rule edited at run time are used. This is a fail-safe mechanism so that a new version of the process will be able to run even when previously edited (and no longer compatible) rules are present.

**Auditing and Rules**

Every use of a business rule can be audited. You can control whether or not the evaluation of the rule will be visible in the audit trail.

Dynamic business rules can be evaluated from a business rule transition or from code. The following table describes how to handle each case:

Rule Evaluated In	To Enable Auditing
Business Rule Transition	In the activity where the business rule transition originates (the <i>from</i> activity), the <b>Generate Events</b> option must be set.
PBL Code	In the activity which causes the execution of the PBL code, the <b>Generate Events</b> option must be set.

**Defining a Business Rule**

The following task outlines the procedures for creating and editing Business Rules.

To create a business rule, your project must have either one or more project variables, or one or more business parameters.

To define a business rule:

1. Right-click on the project where you want to define Business Rules.
2. Select **Business Rules** (💰).  
The Business Rules editor appears.
3. Click the **Add** icon (⊕).  
The **New Business Rule** dialog box appears.
4. Enter the name of your new business rule, and click **OK**.  
The new Business Rule appears in the table.
5. To edit the business rule, click on the Open icon (🔓), or double-click on the business rule name.  
The business rule editor page for that rule appears.
6. By default, the business rule editor page is in simple editor mode. You can also edit in advanced editor mode. Go to the corresponding task listed below for instructions in the mode you will use.
7. To close the business rule editor and all open business rules, click the **X** on the top tab. To close only the page with the business rule you have just saved, click the **X** on the bottom tab.

**Simple Editor Mode**



Use the simple editor mode to define simple business rules without having to write any PBL code.

To edit a business rule:

1. To add a condition, select one of the available project variables or business parameters from the **Add Condition** drop-down list.
2. Click the **Add** button (⊕).  
A new condition line is added.
3. Select a comparison operator from the first drop-down list.

The following table shows the full set of possible operators, but only a subset of these will be available based on the data type of the variable or parameter you are comparing:

Operator	Equivalent PBL Expression
Equals	$x = a$
Not Equals	$x \neq a$
Greater Than	$x > a$
Less Than	$x < a$
Greater Than or Equals To	$x \geq a$
Less Than or Equals To	$x \leq a$
Between	$(x \geq a \text{ and } x \leq b)$
Not Between	$(a > x \text{ or } x > b)$

- Enter the value that will be compared to in the field on the right. Special options can appear for some data types. For example, a time value field will be accompanied by a calendar tool to help you pick a date. If you are using the *Between* or *Not Between* operators, you will need to enter two values.
- Repeat steps 1 through 4 to add more conditions as required.
- If you wish to remove a condition, click the **Remove** button () next to it.
- Once you have set all the conditions in your business rule, save your changes by clicking the Save button () or by clicking **File ► Save** from the main menu.

### Advanced Editor Mode

Use the advanced editor mode to write complex business rules. This mode requires PBL coding, so to use it you should be familiar with PBL syntax.



**Note:** If you write a complex business rule in the advanced editor mode, you will no longer be able to edit this rule in the simple editor mode.

To use the advanced editor:

- In the simple editor, click **Switch to Advanced Editor**.  
The advanced editor opens. This is a PBL code editor. If you have already defined any conditions, these will be shown in code form.
- In the editor, enter any PBL code you require to implement the business rule. You must exit your code with a `return` statement which returns a boolean value (`true` or `false`).  
For more information about PBL, consult the [Process Business Language \(PBL\)](#) on page 252.
- If you want to switch back to the simple editor mode, right-click anywhere on the advanced editor and click **Switch to Simple Editor**.  
Recall that you will not be able to do this if you've edited the rule in such a way that it can no longer be represented in the simple editor.
- Save and close when you are done.


## Letting Participants Edit Business Rules

You can enable participants to edit business rules at run time using the Workspace web application. To do this, you add a global interactive activity with an "Edit Business Rules" implementation type.

To complete this task, at least one role must be defined.

When you provide access to the business rules editor by adding a global interactive activity to a role, you give every participant in that role the ability to edit the business rules selected in that activity.

To add a Global Interactive of type "Edit Business Rules":

1. Insert a global interactive activity () in the role you wish to enable to edit business rules. The **Activity** dialog box appears.
2. Enter a name for the activity in the **Name** field, and click **OK**.
3. Right-click on the activity you have just added, and click **Main Task**. The **Main Task** dialog box appears.
4. In the Implementation Type section, select *Edit Business Rules* from the drop-down list. A list of available business rules appears.
5. To enable WorkSpace editing of a business rule, click on the right side column so it says **Yes**. Alternatively, you can click **All** to enable all business rules, or **None** to disable them. You must enable at least one business rule.
6. When you have enabled the desired business rules for this activity, click **OK**.

The global interactive activity is now set to edit business rules, and will appear in the **Applications** pane of any WorkSpace participant in the role where you added the activity.

## Handling Exceptions

### Exception Handling in Oracle BPM

Oracle BPM provides multiple ways of handling exceptions that occur outside of the normal flow of a program. The specific way used depends on where the exception occurs and what causes it.

Within Oracle BPM exceptions can be classified according to the following distinctions:

- System versus Business Exceptions
- Code-level versus Process-level Exceptions

#### System Versus Business Exceptions

This distinction defines the nature of the exception.

A system exception occurs when there is a problem with one of the components used by a process. These components can include databases, network connections, etc. System exceptions are included in the catalog as part of the standard Java exceptions.

A business exception is designed as part of a process business process, but is something that occurs outside of the normal flow of a process. This allows you to create cleaner processes where the main flow follows the normal use cases. Business exceptions are defined as BPM objects within the catalog.

Another major differences between system and business exceptions is that business exceptions do not roll back activity transactions. This is because business exceptions are considered as a normal part of the process design rather than an error.

#### Code-level Versus Process-level Exceptions

This distinction defines where an exception is handled. All exceptions originate at the code level. However, they can be handled at either the code or process level depending on the requirements of your process design.

Code-level exception handling occurs within the scope of a PBL script. Code-level exceptions handling allows you to write code that directly accounts for the exception within the PBL task where it occurred.

Process-level exception occurs as part of the process design. When an exception occurs that is not explicitly handled within the code, it is propagated up to the process level. Process-level exceptions are handled within in exception flows.

### Cataloging Exceptions

All exceptions are stored in the Catalog. System exceptions are stored as standard Java exceptions. Business Exceptions are stored within a user-defined BPM Objects.

### Example Project

An example project is included which shows different usage scenarios of Exceptions. It is located at `<ORABPM_HOME>/samples/advanced/ExceptionHandling.exp`.

## System Exceptions

System exceptions are those caused by failures on the underlying software or hardware infrastructure. System exceptions are not expected problems within the business process logic.

System exceptions are often caused by temporary errors such as network connectivity failures. These are often temporary errors that can be resolved by re-trying the failed transaction.

All System exceptions have a corresponding Java exception that is included as part of the component catalog, within the standard Java module.

### Transactions

If a System exception occurs during the execution of a process Activity (and it is not handled within PBL code) it causes the Activity transaction to fail. The Process Execution Engine rolls the transaction back and any changes made to process instance variables are lost.

### Retries

When a transaction fails due to System exception, the Process Execution Engine retries the transaction. The number of retries and amount of time in between tries is a configurable property of the Engine.

If the transaction keeps failing after the maximum number of tries, the process instance is routed to the exception handling flow defined for that type of exception.

### Interactive Activities

If a System exception is raised during the execution of an interactive activity, an error message with the exception details is presented to the end user.

### Engine Exceptions

Special exceptions raised internally by the Process Execution Engine, such as internal database time outs and Execution Aborted exceptions, cannot be caught within PBL code or as part of a exception handling flow.

## Business Exceptions

Business exceptions are expected conditions that prevent a process instance from advancing in the process. Unlike System exceptions, business exceptions are triggered by your business process rules. They denote that some condition in the business logic has not been met. Failure to pass a credit score check, for example, could be handled as a business exception.

Business exceptions allow you to create cleaner processes and allow you to define exceptional situations separately from the *happy path* of the process.

Business Exceptions must be defined as BPM Objects within the Catalog. They behave like other BPM Objects and can contain methods, attributes, and presentations. They can be used anywhere within a process.

## Transactions

Business exceptions are expected business situations. Unlike System exceptions, Business exceptions do not cause the running transaction to fail.

If your code throws a Business exception during the execution of a process Activity (and it is not handled within PBL code) it causes the execution of the Activity to exit but the transaction is finished successfully. The Process Execution Engine commits the transaction and any changes made to process instance variables are persisted as usual.

## No Retries

If the execution of a process activity finishes due to a Business exception, the Process Execution Engine does *not* retry the execution of the activity because the execution did not fail.

In general, business exceptions are not resolved by retrying. For example, you may raise a business exception if the credit score for a credit card application is low. Trying the credit check again does not make the score higher.

## Interactive Activities

If a Business exception is raised during the execution of an interactive activity, the interaction window is closed and the work item is no longer on the user work list. The process instance is moved to the exception handling flow.

## Throwing a Business Exception

Within PBL code, you can raise a Business Exception using the `throw` keyword as in the following example:

```
if creditScore > 700 then
  lowScoreEx as LowScoreException = LowScoreException()
  lowScoreEx.value = creditScore
  throw lowScoreEx
end
```

## Code-level Exception Handling

Code-level Exception handling allows you to write code to deal with problems that occur within the scope of a PBL task.

PBL uses the following block structure to handle exceptions:

```
do
  // regular code

on Exception
  // exception handling code

on exit
  // clean-up code (always executed)

end
```

The `do-end` block, though not required, allows you to clearly define the scope of an exception within your code. This is particularly important if you need to nest exceptions within multiple `do-end` blocks.

The `on Exception` statement allows you to define the block of code that executes only if the exception is raised.



The `on exit` statement allows you to perform any clean up operation and releasing resources. This can include cleaning up temporary systems files or database table created within your PBL code. This block is always executed, whether an Exception was raised or not.

### Exception Handling Example.

The following code example demonstrates the syntax for using the `on Exception` and `on Exit` constructs.

```
on TooCloseToDueDateException do
  sendAlert project
    using mailSubject = project.name + "is getting too close to due date",
      mailMessage = "The project " + project.name + "is getting to close to
due date."

on OverdueException do
  sendAlert project
    using mailSubject = project.name + "is overdue",
      mailMessage = "The project " + project.name + "is overdue."

on exit do
  sendAlert project
    using mailSubject = "Work on project " +project.name + " was started.",
      mailMessage = "Work on project " + project.name + " was started on " +
project.startDate
      + ". The project leader for is: " + project.projectLeaderId

end
```

This example uses the default PBL programming style. Other programming styles use different keywords for exception handling, but the underlying concept is the same. If you are using the Java programming style, this is implemented within a try-catch block.

See [Compound Statement](#) on page 323 for more information.

### Propagating Exceptions to the Process Level

In many cases, you may not want to catch exceptions within PBL code. Within your process it may make more sense to let them propagate as process-level exceptions. Any exceptions not explicitly handled within PBL code (with `on Exception` blocks) are propagated. If an exception cannot be completely resolved within PBL code, it should propagate up as a process-level exception, to be handled with an exception handling process flow.

## Process-level Exception Handling

Process-level exceptions occur when a code exception is raised and is not handled within PBL code. The exception is then propagated to the process level.


When a process-level exception occurs, it prevents the normal flow of the process instance from continuing. The process instance gets into a special state and is routed to the exception handling flow defined for that type of exceptions.

## Typical Exception Handling Flow

The following list demonstrates how exception handling occurs within a typical process:

1. The Process Execution Engine begins executing the process instance.
2. An exception occurs within an Activity at the code level.
3. If exception handling code is available, then that code is executed.

The Activity completes successfully, transactions are committed, and the Process Execution Engine continues with the next Activity in the instance.

4. If no exception handling code is available, then the Process Execution Engine cannot continue. The exception is propagated to the Process level
5. At the process level, the following options are possible:
  - If no exception flow has been defined, the instance continues directly to the End Activity and the STATUS predefined variable is set to ABORTED state.
  -  **Note:** When designing your business process, you should always create at least one default exception handling flow.
  - If an exception flow has been defined, the process instance continues through the exception handling flow. If you have defined different exception transitions, the flow is routed through the appropriate transition.
6. Process flow continues through each Activity in the Exception flow.
7. In the Final Activity of the Exception Flow, the Process Execution Engine evaluates the ACTION predefined variable to determine where to continue the flow of the instance.
8. Based on the value of the ACTION predefined variable, instance flow is returned to the main process flow.

## Creating an Exception Flow in a Process

1. Create a new activity outside of your main process flow.  
This activity will be part of your exception flow. For some activity types, you must use control-click to add an activity outside of the main process flow.
2. Right-click within you process and select **Add Exception Transition To**
3. Select the Activity created above
4. On the Description tab enter a name for the exception transition
5. Select the **Properties Tab**.
6. Choose an Exception Name from the drop-down list
7. Choose an instance variable
8. Click **OK**

## Creating a Business Exception

Before performing this procedure, ensure that you have created a module where you want to create a Business Exception component. See [Creating a Module](#) on page 152.

To create a Business Exception component:

1. Right-click on the module where you want to create a Business Exception.
2. Select **New ► Business Exception**
3. Enter a new for the new component and click **OK**.

## Business Activity Monitoring (BAM)


The following topics describe the Business Activity Monitor and provide information on the BAM database, creating BAM Dashboards, and configuring BAM in Oracle BPM Studio.

## BAM Overview

Business Activity Monitoring (BAM) allows you to store, analyze, and display statistics about your business process execution.

BAM provides information about process instance performance and process workload. This information can be used to present almost real-time business processes metrics. You can then use these to analyze and then improve or adapt business processes based on real-world conditions.

To store and present this information, BAM contains the following:

Database	BAM data is stored within a database. In Oracle BPM Studio, this information is stored internally as part of the embedded process execution engine database. In Oracle BPM Enterprise, you must configure an external database to function as the BAM database.
SQL Queries	You can write queries that access the information stored in the BAM database. These queries are contained within a BPM Object Method. When you create a BAM Dashboard using the wizard, the wizard automatically creates queries based on the type of Dashboard template you choose. You can customize these queries or create your own queries and dashboards to customize the way you present BAM.
BAM Dashboards	BAM Dashboards allow you to display BAM information in a meaningful and useful way. BAM Dashboards also allow you to drill down from a general view of a process to more specific information such as an order or claim.   <b>Note:</b> BAM Dashboards require the Flash Plugin.

## Enabling and Configuring BAM in Studio

BAM is included within Oracle BPM Studio. This allows you to test real-time dashboards using BAM data generated by the embedded Process Execution Engine.

By default, Oracle BPM Studio does not generate BAM data. The following procedures show you have to enable BAM and set other BAM properties.

1. Right-click on any project within Studio.
2. Select **Engine Preferences**.
3. Click BAM in the left-hand tree.
4. Click the checkbox next to **Enable BAM**.

You can also set other BAM properties on this page.

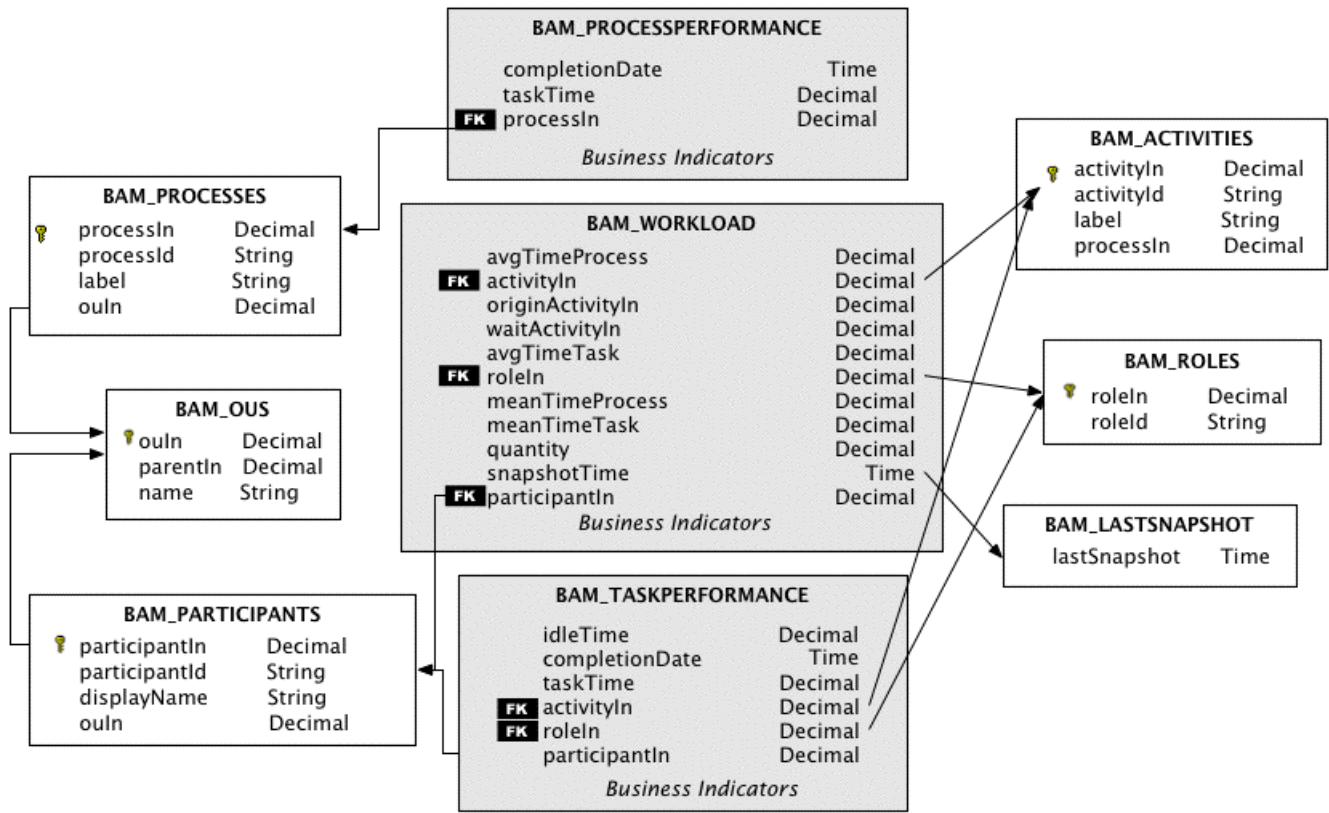
## BAM Database

The BAM database is used to store information about your business processes.

The BAM database stores the following types of information about a process:

1. Workload
2. Task Performance
3. Process Performance

The following diagram shows the relationship between each of the BAM tables:



How BAM Database is Populated

BAM database is populated with the information generated by auditing events. Auditing events generation can be enabled for the whole process, for a subset of activities or for a particular activity. For more information on how to configure auditing events generation please see the *Oracle BPM Enterprise Administration Guide* and the *Oracle BPM Studio User Guide*.

Using Variables in BAM

When creating a Project variable, you can define it as a Business Indicator variable. This allows the variable to be stored in BAM the database.

When you add Business Indicator variable to your process, a column is added to the following BAM database tables: Workload, Task Performance and Process Performance. The name of this column is the Business Indicator name preceded by the prefix "V\_".

If you define a business dimension, the workload table contains one row for each possible value of this business dimension present in the process. Each row shows the quantity of instances that match that business dimension. If the business dimension has a numeric type, the value stored in BAM tables indicates the range that corresponds to the value of the business dimension.

When you define a measurement business variable, the sum of this variable's value for all in flight instances is stored into workload table. If business dimensions were defined as well, then this sum will be divided into as many rows as business dimension values present in flight instances.

Task performance table stores one row for each instance that completes an activity. Each of these rows contains the value of dimensions and measurements at the time the instance completed the activity.

In a similar way, process performance table stores one row for each instance that gets to the end activity. Each of these rows contains the value of dimensions and measurements at the time the instance completed the whole process.

## Creating a Predefined BAM Dashboard

Oracle BPM Studio provides a quick method of creating BAM Dashboards using predefined templates.

To create a Predefined BAM Dashboard:

1. Ensure that you have enable BAM in Studio.  
See [Enabling and Configuring BAM in Studio](#) on page 235 for more information.
2. Right-click on the Project where you want to create a BAM dashboard.
3. Select **Add BAM Predefined Dashboard**.
4. Select the type of template you want to use. You can select more than one template if necessary.
5. Click **OK**.
6. Select the Process whose BAM information you want to report.
7. Select the Role you want to allow to view the BAM information.

Anyone assigned to this role is able to view the BAM dashboard.

Studio creates BAM Dashboards for each of the template you selected. Dashboard Global activities for each dashboard are added to the process.

See [Viewing BAM Dashboards in Studio](#) on page 237 for information on viewing BAM Dashboards in Studio.

## Viewing BAM Dashboards in Studio

After creating a BAM Dashboard, you can view it using Oracle BPM Studio's Process Execution Engine.

Ensure that you have a participant in the process who has been assigned the role where the BAM Dashboard reports are located.

1. Start the Oracle BPM Studio Process Execution Engine
2. Launch WorkSpace.
3. Login with the username of a participant who has been assigned the role used to view BAM information.
4. Click Applications to view the BAM Dashboard reports.

## BAM Database Reference

The BAM database has a star shaped style schema. This reference describes the fact tables and dimension tables in the BAM database.

### Fact Tables:

- BAM\_WORKLOAD
- BAM\_TASK\_PERFORMANCE
- BAM\_PROCESS\_PERFORMANCE

### Dimension Tables:

- BAM\_OUS
- BAM\_ROLES
- BAM\_PARTICIPANTS
- BAM\_PROCESSES
- BAM\_ACTIVITIES

**BAM\_WORKLOAD**

This table contains information about the work items in process. The BAM Updater populates this table with the information it obtains periodically from the engine.

Field Name	Value	NULL Value	Description
snapshot	TIMESTAMP	NOT NULL	The date and time of the snapshot this row belongs to.
activityIn	DECIMAL(10)	NOT NULL	The identification number (IN) of the activity where the work items this row represents, are sitting in. Use this IN in join queries against the BAM_ACTIVITIES table.
roleIn	DECIMAL(10)	NOT NULL	The identification number (IN) of the role where the work items this row represents, are sitting in. Use this IN in join queries against the BAM_ROLES table.
participantIn	DECIMAL(10)	NOT NULL	The identification number (IN) of the participant the work items this row represents, are assigned to. Use this IN in join queries against the BAM_PARTICIPANT table.
origActivityIn	DECIMAL(10)	NOT NULL	If the work items this row represents, are sitting in a subflow activity, this field indicates the identification number of the activity in the subprocess associated to this subflow activity. Otherwise the value of this field is 1.
waitActivityIn	DECIMAL(10)	NOT NULL	If the work items this row represents, are sitting in the sub-process of a subflow activity, this field indicates the identification number of the subflow activity in the parent process. Otherwise the value of this field is 1.
quantity	DECIMAL(10)	NOT NULL	Indicates the number of work items that match the value of the following fields in this row: <ul style="list-style-type: none"> <li>• processIn</li> <li>• activityIn</li> <li>• roleIn</li> <li>• participantIn</li> <li>• business dimension</li> <li>• latsnapshot</li> </ul>
avgTimeTask	DECIMAL(10)	NOT NULL	The average waiting time (in seconds), that the instances represented by this row, spent in the activity at the moment of the snapshot.
meanTimeTask	DECIMAL(10)	NOT NULL	The median of the waiting time (in seconds), that the instances represented by

Field Name	Value	NULL Value	Description
avgTimeProcess	DECIMAL(10)	NOT NULL	this row, spent in the activity at the moment of the snapshot. The average waiting time (in seconds), that the instances represented by this row, spent in the process at the moment of the snapshot.
meamTimeProcess	DECIMAL(10)	NOT NULL	The median of the waiting time (in seconds), that the instances represented by this row, spent in the process at the moment of the snapshot.

**Primary Key:** The primary key constraint is not defined in the database schema. The BAM Updater checks this constraint.

The following fields form the primary key:

- activityIn
- processIn
- roleIn
- participantIn
- snapshotTime
- Business Dimensions

#### Foreign Keys

Foreign Key	Referenced Table
activityIn	BAM_ACTIVITIES
waitActivityIn	BAM_ACTIVITIES
origActivityIn	BAM_ACTIVITIES
roleIn	BAM_ROLES
participantIn	BAM_PARTICIPANTS

#### BAM\_TASKPERFORMANCE

This table contains performance information for every work item that has completed an activity.

Field Name	Value	NULL Value	Description
activityIn	DECIMAL(10)	NOT NULL	The identification number (IN) of the completed activity. Use this IN in join queries against the BAM_ACTIVITIES table.
roleIn	DECIMAL(10)	NOT NULL	The identification number (IN) of the role the completed activity is assigned to. Use this IN in join queries against the BAM_ROLES table.
participantIn	DECIMAL(10)	NOT NULL	The identification number (IN) of the participant the completed activity is assigned to. Use this IN in join queries against the BAM_ROLES table.



Field Name	Value	NULL Value	Description
completionDate	TIMESTAMP	NOT NULL	The time when the work item completed the activity. This time is stored in GMT-0.
taskTime	DECIMAL(10)	NOT NULL	The time (in seconds) the work item took to complete the activity.
idleTime	DECIMAL(10)	NOT NULL	The time the work item waited until its first execution.

#### Foreign Keys

Foreign Key	Referenced Table
activityIn	BAM_ACTIVITIES
roleIn	BAM_ROLES
participantIn	BAM_PARTICIPANTS

#### BAM\_PROCESSPERFORMANCE

This table contains performance information for every work item that has completed the whole process.

Field Name	Value	NULL Value	Description
processIn	DECIMAL(10)	NOT NULL	The identification number (IN) of the process. This number identifies the process in the organizational unit. It may vary between deployments. Use the IN only for join queries with other tables. Do not use the IN directly in your queries, instead use the processID.
completionDate	TIMESTAMP	NOT NULL	The time when the work item completed the process. This time is stored in GMT-0.
taskTime	TIMESTAMP	NOT NULL	The time (in seconds) the work item took to complete the activity.

#### Foreign Keys

Foreign Key	Referenced Table
processIn	BAM_PROCESSES

#### BAM\_LASTSNAPSHOT

This table stores the time when the BAM Updater took the last snapshot. The BAM Updater populates the Workload with the data obtained in each snapshot.

Field Name	Value	NULL Value	Description
lastsnapshot	TIMESTAMP	NOT NULL	The time of the last snapshot. Use this time to obtain from the Workload table the rows that correspond to the most up-to-date data.



**BAM\_OUS**

Field Name	Value	Null Value	Description
ouIn	DECIMAL(10)	NOT NULL	The identification number (IN) of the organizational unit. This number identifies the organizational unit in its parent organizational unit. It may vary between deployments. Use the IN only for join queries with other tables. Do not use it directly in your queries, instead use the name.
parentIn	DECIMAL(10)	NOT NULL	The identification number (IN) of the parent organizational unit. If this row corresponds to the Organization the value of this field is -1.
name	STRING(255)	NOT NULL	The name of the organizational unit. Use this ID in where clauses to restrict the query to a certain role.
fullPathName	STRING(512)	NOT NULL	The complete name of the organizational unit, which includes the complete name of its parent organizational unit.

**Primary Key:** ouIn**BAM\_ROLES**

Field Name	Value	NULL Value	Description
roleIn	DECIMAL(10)	NOT NULL	The identification number (IN) of the role. This number identifies the role in the organizational unit. It may vary between deployments. Use the IN only for join queries with other tables. Do not use the IN directly in your queries, instead use the roleID.
roleID	DECIMAL(10)	NOT NULL	The ID that identifies the role in the organizational unit. Use this ID in where clauses to restrict the query to a certain role.

**Primary Key:** roleIn**BAM\_PARTICIPANTS**

Field Name	Value	NULL Value	Description
participantIN	DECIMAL(10)	NOT NULL	The identification number (IN) of the participant. This number identifies the participant in the organizational unit. It may vary between deployments. Use the IN only for join queries with other tables. Do not use the IN directly in your queries, instead use the participantID.

Field Name	Value	NULL Value	Description
participantID	STRING(255)	NOT NULL	The ID that identifies the participant in the organizational unit. Use this ID in where clauses to restrict the query to a certain participant.
ouIn	DECIMAL(10)	NOT NULL	The identification number (IN) of the organizational unit the participant belongs to. Do not use this number directly in your queries, use it only for join queries against the BAM_OU table.
displayName	STRING(255)		The localized, human readable name of the organizational unit. The system locale settings of the environment where the BAM Updater runs determine the localization of this field.

**Primary Key:** participantIn

**Foreign Keys**

Foreign Key	Referenced Table
ouIn	BAM_OUS

#### BAM\_PROCESSES

Field Name	Value	NULL Value	Description
ouIn	DECIMAL(10)	NOT NULL	The identification number (IN) of the organizational unit where the process is deployed. Do not use this number directly in your queries, use it only for join queries against the BAM_OU table.
processIn	DECIMAL(10)	NOT NULL	The identification number (IN) of the process. This number identifies the process in the organizational unit. It may vary between deployments. Use the IN only for join queries with other tables. Do not use the IN directly in your queries, instead use the processID.
processId	STRING(255)	NOT NULL	The ID that identifies the process in the organizational unit. Use this ID in where clauses to restrict the query to a certain process.
label	STRING(255)	NOT NULL	The localized, human readable name of the process. The system locale settings of the environment where the BAM Updater runs determine the localization of this field.

**Primary Key:** processIn .

**Foreign Keys**

Foreign Key	Referenced Table
ouIn	BAM_OUS

**BAM\_ACTIVITIES**

Field Name	Value	NULL Value	Description
activityIn	DECIMAL(10)	NOT NULL	The identification number (IN) of this activity. This number identifies the activity in the organizational unit. It may vary between deployments. Use the IN only for join queries with other tables. Do not use the IN directly in your queries, instead use the activityID.
activityId	STRING(255)	NOT NULL	The ID that identifies the activity in the organizational unit. Use this ID in where clauses to restrict the query to a certain activity.
processIN	DECIMAL(10)	NOT NULL	The identification number (IN) of the process this activity belongs to. Do not use this number directly in your queries, use it only for join queries against the BAM_PROCESSES table.
label	STRING(255)		The localized, human readable name of the activity. The system locale settings of the environment where the BAM Updater runs determine the localization of this field.

**Primary Key:** activityIn

**Foreign Keys**

Foreign Key	Referenced Table
processIn	BAM_PROCESSES

## Unit Testing BPM projects

The following topics describe how to create and run Process (PUnit) and Component (CUnit) test cases.

### Unit Test Overview

A unit test is a piece of code used to test pieces of code. Oracle BPM Provides a framework for testing individual BPM Components or an entire Process.

The Oracle BPM unit tests are based on the JUnit unit test framework. See <http://www.junit.org> for more information.

Oracle BPM provides two types of unit test suites:

Unit Test Type	Description
CUnit Test	Allows you to create unit tests for individual BPM Objects. New CUnit test suites are created with a

Unit Test Type	Description
PUnit Test	<p>default method. You can add other methods as necessary.</p> <p>Allows you to create a test framework for an entire Process. New PUnit test suites are created with a default, setUp, and tearDown method.</p>

Within Oracle BPM, unit tests behave like other BPM Objects. They can contain Attributes, Groups, Presentations, and Methods.

### Example Project

An example project is included which shows usage of PUnit and CUnit test cases. It is located at `<ORABPM_HOME>/samples/advanced/BPMUnitTestExample.exp`.

## Creating a Unit Test

The following procedure shows you how to create CUnit and PUnit test suites.

1. In the Project Navigator, right-click on the Module where you want to create a Unit Test.
2. Select **New ► PUnit Suite** or **New ► CUnit Suite**
3. Enter a name for your test suite.
4. Enter the destination module.
5. Click **Ok**.

The new test unit suite appears as a resource under your Module. You can now define your unit test within the Methods of your test suite.

## Running a Unit Test

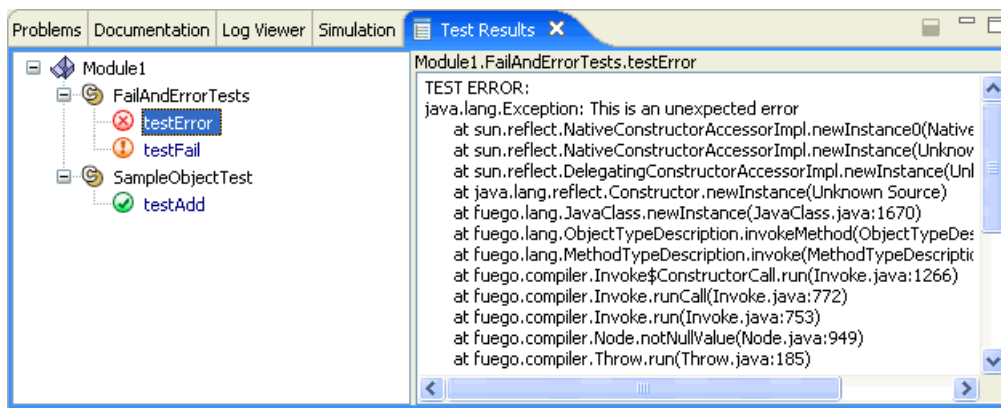
After you have created unit tests for your Components or Project, you can run the tests within Oracle BPM Studio. Unit test results are displayed in the Test Results View.

To run a unit test:

1. Right-click on the Module or unit test Component.
2. Select **Run Unit Test**.  
The test results appear in the Test Results View.
3. Select the test in the left-hand pane.

## Test Results View

The Test Results View displays results from PUnit and CUnit tests.



## Correlations

In Oracle BPM, a correlation is an association between a set of one or more values and a process instance. With correlations you assign a natural key to uniquely identify a process instance.

Consider a situation where you have a process that manages purchases. Your suppliers can access the system through some kind of interface, perhaps a Web front end. You create an instance in your process by sending a purchase order to a supplier. Your process will expect the seller to return an acknowledgment for the order, stating if the order is accepted or not, so the process waits for this acknowledgment.

For the process to continue, the acknowledgment must be routed to the same instance that originated the purchase order. However, your supplier does not have the instance ID. So then the question is: How can you route the acknowledgment from the supplier to the correct instance? You could send the instance ID to your supplier, but this ID has no meaning to the supplier, and the supplier may not even have a way of storing it. In fact, the instance ID has no meaning anywhere outside the process.

In this situation, you can define a *correlation set* that correlates a *business token* to a given instance. A business token is simply a value or set of values that has business meaning and is also unique to each instance. In this example, you can use a purchase order number as a business token, but you can't use the date or amount of the purchase, because these values are not unique to a given order.

## Correlation Sets

You create correlation sets at design time. A correlation set has a name and one or more correlation properties. Each property is a reference to an argument of the argument set.

### Correlation Sets

You can have one or more correlation relationships. For example, you may want to map a purchase order number to more than one system. This would be the case if you have a supplier and a separate shipping service. You may also have more than one process in your project that can use the same business token.

For each relationship, you create a *correlation set*. A correlation set is composed of one or more *correlation properties* that uniquely identify the instance.

## Defining a Correlation Set

To define and use correlation set, you must first create the set, then add properties to it, and then map these properties to process arguments.

Before you define a correlation, you should know the name and data type for each of the properties you will add to the correlation. The following steps assume that you have a process design open in the editor.

To define a correlation set:



1. [Creating a Correlation Set](#) on page 246.
2. [Adding Correlation Properties](#) on page 246.

### Creating a Correlation Set

You create a correlation set from the **Argument Mapping** dialog box. You can define argument mappings in the Begin, Subflow, Process Creation, Termination Wait, Process Notification, and Message Wait flow objects.

Before you define a correlation, you should know the name and data type for each of the properties you will add to the correlation. The following steps assume that you have a process design open in the editor.

To define a correlation set:

1. In the process design editor, right-click on the Begin activity and click **Argument Mapping**.  
The **Argument Mapping** dialog box will appear.
2. Click on the **Correlations** Icon ().  
The **Correlations** dialog box appears.
3. Click the **Add** button () in the **Available Correlations** section.  
The **New Correlation Set** dialog box appears.
4. Specify the **Correlation Set Name**, and click OK.  
The new correlation set is created, and added to the **Available Correlations** list.

### Adding Correlation Properties

Once you have defined a correlation set you must add properties to it.

To perform this task, you need to have created at least one Correlation Set. The task assumes that the **Correlations** dialog box is open.

To add a correlation property:

1. In the **Available Correlations** list of the **Correlations** dialog box, right-click on the name of the Correlation set you want to add a property to, and click **Add Property**.  
The **Property** dialog box appears.
2. In the **Property** dialog box, enter a name for the property.
3. Select a [Correlation Property Data Types](#) on page 246 from the **Type** drop-down list.
4. Click **OK**.

## Correlation Property Data Types

Correlation Properties can be defined with one of several data types, as described in this section. Complex data types, such as objects or arrays, are not allowed.

A correlation property can be one of the following data types:

- Bool
- Int
- Real
- Time
- Decimal
- String

## Correlations Example

Oracle BPM includes an example project and an example Java application that sends notifications to process instances of the process using a correlation property.

## The CorrelationsExample Project

The CorrelationsExample project contains the Accept Invoice process, its associated screenflows. Each component is described below.

You can find the CorrelationsExample.exp file the OraBPMStudioHome\samples\advanced\correlations folder.

### Project Elements

The following table lists additional elements of the CorrelationsExample project.

Name	Type	Description
<a href="#">The Accept Invoice Process</a> on page 247	process	The main process of the project. This process is where the correlation is implemented. Click on the link for a full description.
Invoice Input	screenflow	Screenflow called by the Generate Invoice activity. You use it to create a new invoice.
Release Invoice	screenflow	Screenflow called by the Release Invoice activity. You use it to specify the number of the invoice you want to release.

### The Accept Invoice Process

The Accept Invoice process is a skeleton version of a process designed to be used in an accounts payable department for initial invoice processing.

The primary purpose of this process is to demonstrate correlations, so it implements only two data attributes, the invoice number (`invoiceNr`), and the approval status (`approvalStatus`). It uses one correlation set (Invoice) with one attribute (number).

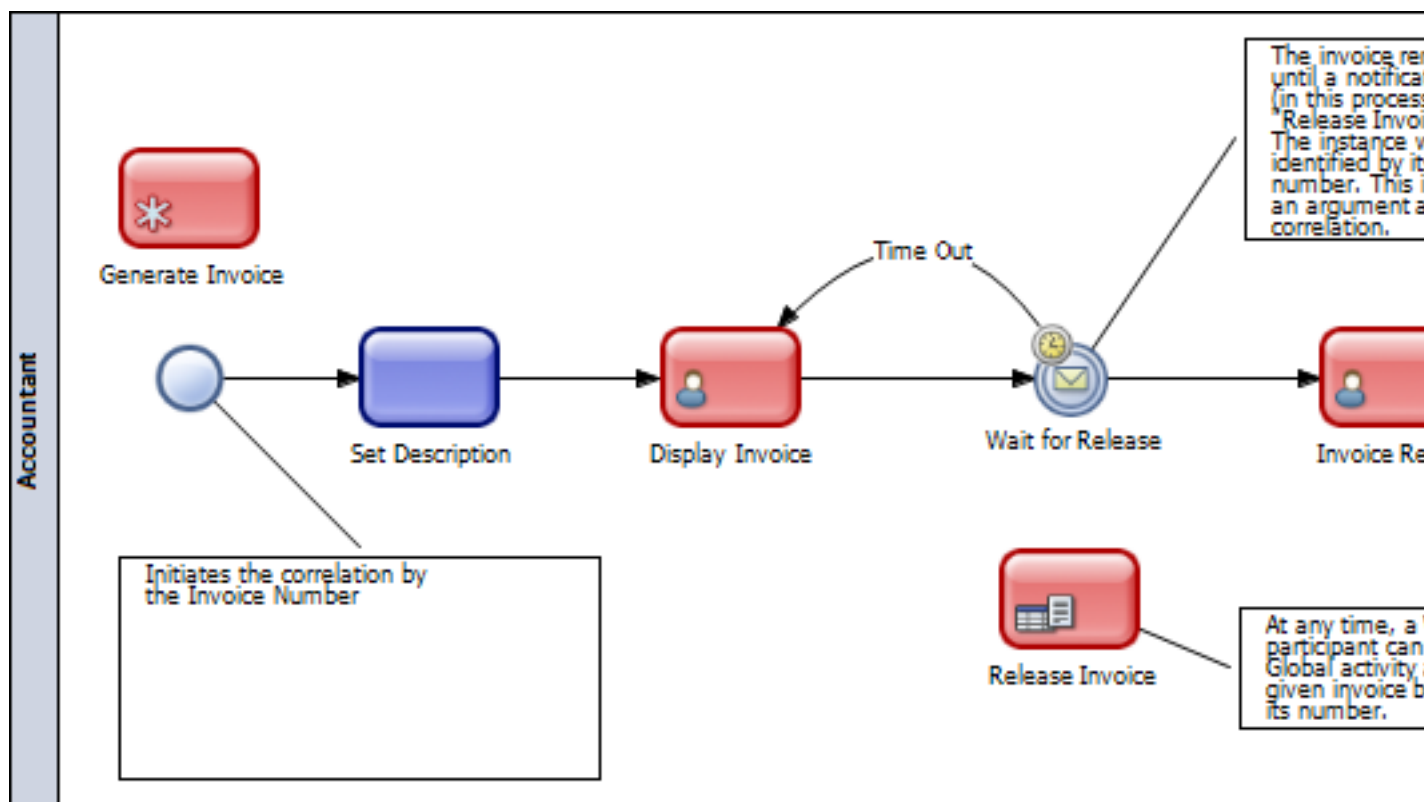


Figure 5: The Accept Invoice Process

## Process Flow

The process works as follows:

- You create a new invoice with the **Generate Invoice** activity. The only data you must input is an invoice number.
- The **Set Description** activity sets the instance description (`this.description`) so that it contains the invoice number.
- You display the invoice at the **Display Invoice** activity. The main purpose of this activity is to have an interactive activity before the **Message Wait**, described below.
- The **Wait for Release** activity (which is a **Message Wait** activity) holds the instance till one of two things happen:
  - It receives a notification.
  - The time out interval of the due transition elapses (it's set at two hours).

You "release" the invoice by sending a notification. To do this, you can use the **Release Invoice** activity, which is listed in the **WorkSpace Applications** panel, or you can release it from the included **ReleaseInvoice** Java application. If you release it from the **Release Invoice** activity, the `approvalStatus` process variable is set to "Released". If you release it from the Java application, `approvalStatus` is set to "Approved".

- Upon release, the instance proceeds to the **Invoice Ready** activity. When you execute this activity, **WorkSpace** displays the status of the invoice.

## Running the Example Process

Follow these steps to run the correlations example project.

1. Import the **CorrelationsExample** project.

The project is located at

`<ORABPM_HOME>/samples/advanced/correlations/CorrelationsExample.exp`.

2. Start the project.

See [Running a Project in Studio](#) on page 33.

3. Launch **WorkSpace** and login with user `test`.

4. Click on **Generate Invoice** application link.

5. Enter any integer number in the **Invoice Number** field and click **OK**.

This creates a new process instance, using the invoice number as a correlation value to identify it. You should see the instance in your **Work Items** list.



**Important:** Every time you generate a new Invoice instance, you must specify a different number. Otherwise, you get an error message because correlation values must be unique.

6. Click on **Display Invoice** to confirm the information of your new instance.

Now the process instance is sitting on the **Wait for Release** activity of [The Accept Invoice Process](#) on page 247.

7. Send a notification to the instance, identifying it by its invoice number.

You have two ways of notifying the process instance:

- Click on the **Release Invoice** application link, enter the invoice number of the instance you want to notify and click **OK**.
- Use the accompanying Java program **Release Invoice**. This program simulates an external system sending a notification to your BPM process. See [Running Correlations Java program](#) on page 249 for details.

The process instance you notified should now appear back in your list of **Work Items**. The instance is now sitting on the **Invoice Ready** activity of [The Accept Invoice Process](#) on page 247.



8. You may now click on **Invoice Ready** to confirm the right instance was notified.

If you used the **Release Invoice** activity to notify the instance on the previous step, you should see a message saying the instance was **Released**. If you used the external Java program, you should see a saying the instance was **Approved**.

### Running Correlations Java program

Follow these steps to run the accompanying Java program which complements the CorrelationsExample project.

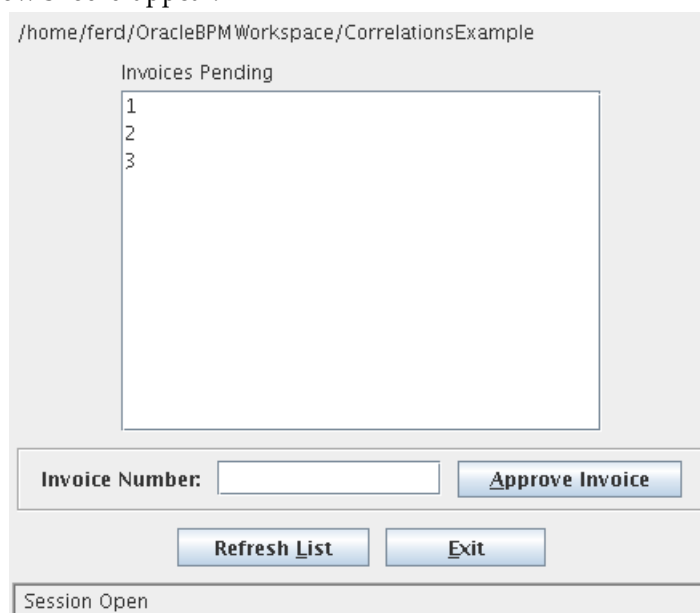
Before running this Java application, the CorrelationsExample project must be running in Studio. See [Running the Example Process](#) on page 248.

If you are interested in learning how the example is implemented, the complete Java code is located at <ORABPM\_HOME>/samples/advanced/correlations/java/.

1. Run script run.bat (Windows) or run.sh (Linux/MacOSX) to start the ReleaseInvoice Java program.

The script is located at <ORABPM\_HOME>/samples/advanced/correlations/.

The application window should appear.



**Figure 6: ReleaseInvoice example Java application**

It connects to the running process and retrieves the list of process instances waiting on the **Wait for Release** activity. See [The Accept Invoice Process](#) on page 247.

2. Select any invoice number from the list and click **Approve Invoice**.

A confirmation message appears. Click **OK**. The selected invoice number show no longer appear on the list.

3. Login to WorkSpace to verify that the associated process instance in the [The Accept Invoice Process](#) on page 247 has been notified and approved. See [Running the Example Process](#) on page 248.

## End-User Interfaces on Oracle BPM

Most Oracle BPM processes include one or more interactive activities, meaning activities which are carried out by people who are participants in the process. For people to interact with an activity, you will need to build a user interface.

End users interact with Oracle BPM through a Web application. This can either be the WorkSpace application which is part of the BPM suite, or a custom application that interfaces to the process execution engine through the Process API, or PAPI.

This section covers development using WorkSpace. PAPI-based development is covered separately.

### The WorkSpace

End-users interact with a BPM process through the WorkSpace, which is a web application. Hence, participants interact with web pages that can contain a form to complete or data for the participant to work with. See [WorkSpace documentation](#).

In WorkSpace, you can create a user interface for a given form or data display by creating an Oracle BPM *presentation* or by writing a JSP page.

Both methods are used to create a web page or form which the participant will use to interact with the data. Presentations can be built quickly in Studio using the Presentation Wizard, while JSP pages can implement functionality and have a look and feel precisely defined by developers. Even if the process you are developing will be deployed with JSP pages, you can use presentations to have something working quickly.

### BPM Objects

In order to build a presentation, you need a BPM Object with attributes. In effect, a presentation is actually a UI layer built on top of the data model of the object. Hence, your first step will be to define the BPM Object and its data model. See [BPM Objects](#) on page 189 for a detailed description.

### Presentations

A presentation is actually an element of the BPM Object, and a single BPM Object can have more than one presentation. You can define several presentations to the same BPM Object when you do not want to show every attribute to every participant. Also, in a presentation you can set whether an attribute field can be edited or is read-only. Therefore, two presentations on the same object could show identical data, but the fields available for editing will be different, as they would be intended for different participants.

### Screenflows

Presentations can be used directly from the process, but normally they are used by *screenflows*, which are a specialized kind of process for user interaction sequences. Screenflows are in turn called from the main process. See [Screenflow Overview](#) on page 128.

## Building a User Interface

There is more than one way to build a user interface with Oracle BPM. This section outlines how to build a user interface using a BPM Object with BPM Object Presentations. The presentations are called in turn from a screenflow.

This is a standard Oracle BPM approach to user interaction, and you should be familiar with it even if you will use other methods, such as JSP pages.

1. Define the information which should be present in the user interface, including both the data the user will see and the data the user will input. You should have a unique name for each data element and know what data type best represents it.

Consider the following tips:

- The best data type for a given data element may not always be obvious. For example, it's easy to see that you must use a String data type for a name field, but do you want to use a number for the postal code? What happens if the process will be used in countries where postal codes can include letters?
  - If you will receive or send data to an external application or service, some data type decisions have already been made for you. You can follow them precisely or you may elect to use a different type within your process and convert where data exchange requires it.
2. [Creating a BPM Object](#) on page 191 with attributes for the data you defined in the first step.
    - If you need to work with tabular data, such as a list of items on an invoice, you can use *groups*. You do not need to explicitly represent each item (*item1*, *item2*, *item3*, and so forth.). Instead, you should create a group such as *items[ ]*.
    - For strings, you should set a maximum length even though this is optional. The **Presentation Wizard** will later use this maximum length to determine whether to use a one-line or multiple-line text box when you create the presentation. The default threshold is 20. You can change this value, called the *Maximum Column Count*, in the **BPM Preferences** dialog box, in the **Presentation Preferences > Text** page.
    - Some information may be derived mathematically from other data. Such data should not be stored. For instance, the total value on an invoice is a sum of the other invoice values. You should implement such data elements as *Virtual Attributes* on page 195.
  3. [Creating a Presentation](#) on page 198 for the BPM Object. Design each presentation as a function of a particular task in your process. In some cases, you will be able to use the same presentation from different activities. As a general rule, each presentation should show or allow input of the data required for the activity it is designed for, and no more.
  4. Create a screenflow with an interactive component call activity (🔴). The main task of this activity should be of implementation type BPM Object Interactive Call. Choose the Use BPM Object Presentation option, and specify the name of the presentation you want to use.
  5. Use the screenflow created in step 4 from an interactive activity in your process design.

# Process Business Language (PBL)

---

## PBL Overview

Process Business Language is the programming language used within Oracle BPM projects where code is required to implement process features or to integrate with external resources.

PBL is simple, high-level language which treats components as objects. PBL can be used to define business rules and logic within Activities and certain types of Transitions. The PBL development environment is integrated within Oracle BPM Studio.

This language is specifically designed to integrate systems and to clearly express business process logic. In addition, PBL supports the following features:

- Choice of syntax style, which can be native PBL, Java, or Visual Basic
- Integration with various back-end technologies including COM, CORBA, XML, SQL, Web Services, and Java
- A modern editor that supports syntax coloring, code completion, templates, and real-time error checking
- Component Libraries
- Regular Expressions

Methods compiled from PBL code are published together with the rest of a project for deployment, but a deployed project contains JVM byte code and not PBL source code. Therefore PBL code can only be created, edited, and tested using Studio.

## Language Basics

### PBL Methods

Introduces Business Process and Business Object methods

In Studio, a method can either be a *Business Process* or a *Business Object*. Both kinds support similar features, but they differ in:

- Their visibility
- The sets of available predefined variables they can access
- Their runtime environment

Business Process methods can only be accessed from the process to which they belong. They are usually the implementation of an Activity. They have several process-related, predefined variables and they always execute inside a process-controlled transaction.

Business Object methods can run on the server side or the client side. They can be defined as functions and inherit behavior from a superclass of the object that contains them. Typically, they are visible from the entire project.

### Comments

Describes purpose and syntax of comments

Comments are text notes which can be read by humans but are ignored by the compiler. Including comments in your code helps make it easier for you and others to read it. Comments are especially useful to record the

intent of a particular piece of code, since it may not be clear to others or, after a period of time, to the original programmer. That said, code readability is not achieved exclusively by including comments. It is also enhanced by adhering to coding conventions and using explicit variable and object names.

Under PBL, comments can be single line or multi-line, and are delimited either with standard C++/Java syntax in the Java and PBL Styles, or with Visual Basic syntax in the Visual Basic style, as described below.



**Tip:** Always remember that when you write comments, you should explain the *why* and not the *how*. The how can be read from the code itself.

### PBL and Java Style Comments

Single line comments are denoted with a double forward slash (//):

```
//This is a single line comment.
```

Multi-line comments are enclosed between a forward slash and an asterisk (/\*) and an asterisk and a forward slash (\*/):

```
/* This is a multi-line comment. It can span multiple lines
of code and can be as long as you want it to be.
You do not have to worry about line breaks, although
you may add them if you want to. */
```

### Visual Basic Style Comments

Visual Basic style uses a single apostrophe (') for single line comments:

```
' This is a comment in the Visual Basic style.
```

## Expressions

Expressions are operations in algebraic format that yield a value when evaluated.

An expression consists of *operators* and *operands*. Operators are special symbols commonly used in expressions, denoting the operations to be performed with the operands they are adjacent to. Operands can be variables or function invocations which return a value that can be operated on by the relevant operators.

Expressions must operate on compatible variable types. Type checking is performed at compile time to guarantee that no runtime errors occur due to an invalid mix of types. Some expression examples follow.

Expressions with numerical values:

```
//Variable c is assigned the sum of a and b.
c = a + b
```

```
// myVariable is assigned the product of 12 by the sum of yourVariable and
ourVariable.
myVariable = 12 * (yourVariable + ourVariable)
```

Expressions with string values:

```
employeeName = firstName + " " + lastName
```

### Precedence

Notice the parentheses in the expression example above. Parentheses play a role in *precedence*. Precedence is the order in which operations take place in a mathematical equation. This is sometimes called *order of operations*. Any operation inside parentheses is evaluated first, and then followed by other operations.

If you evaluate the example:

```
myVariable = 12 * (yourVariable + ourVariable)
```

using 10 for yourVariable and 5 for ourVariable, the order of the operation is the following:

1. yourVariable is added to myVariable resulting in 15.

2. 15 is multiplied by 12 resulting in 180.
3. myVariable is assigned the value 180.

For further information on the different operators, please see Operators.

### Conditional expressions

Conditional expressions assign a value to a variable depending on the result of an expression. The format of a conditional expression is as follows:

```
<Boolean expression> ? <expression> : <expression>
```

If the Boolean expression evaluates to true, the value to the left of the colon is assigned. If it evaluates to false, the value to the right of the colon is assigned. Now, look the next example.

```
myResult = (show = 1) ? "on" : "off"
```

In this example, the variable myResult is assigned the value "on" if the value of the variable show is equal to 1. Otherwise, "off" is assigned to myResult.

## Programming Styles

Describes PBL, Java, and Visual Basic programming styles

Studio supports different programming styles to reduce the time needed to learn how to program business process methods. Each style mimics a well-known programming language as precisely as possible and adds the features that are required to write your business rules effectively.

Oracle BPM Studio supports the following programming styles:

- PBL: The native Process Business Language (PBL) syntax
- Java
- Visual Basic

All available styles are functionally identical except where specifically noted. In other words, the Java and Visual Basic styles are *not* Java or Visual Basic. They are actually PBL formatted with Java or Visual Basic syntax as a programming aid to people familiar with these languages.

### PBL Programming Style

This is the native and recommended style. Also, most of the programming examples in this documentation are in the native PBL style. The following example shows some of the characteristics of the PBL programming style:

```
firstName as String
lastName as String
selectedButton as String

// Ask the person's name
input "First Name:" : firstName, "Last Name:" : lastName
    using title = "Enter Your Name", buttons = ["Done", "Cancel"]
    returning selectedButton = selection

// Check the button pressed
if selectedButton = "Done" then
    display "Hello " + firstName + "!"
else
    display "Hello!"
end
```

### Java Programming Style

This style emulates Java syntax and adds several features to match PBL expressions. These added features include:

- Output arguments
- Input and display statements
- Variable auto-initialization
- Transformations

The following example shows some of the characteristics of the Java programming style:

```
String firstName;
String lastName;
String selectedButton;

// Ask the person's name
input("First Name:" firstName,
      "Last Name:" lastName, title : "Enter Your Name", buttons : { "Done",
"Cancel" }, out selection : selectedButton);

// Check the button pressed
if (selectedButton == "Done")
{
    display("Hello " + firstName + "!");
}
else
{
    display("Hello!");
}
```

### Visual Basic Programming Style

This style emulates Microsoft Visual Basic syntax. However, unlike Visual Basic, the Visual Basic style is case sensitive. This programming style also has several additional features including:

- Input and display statements.
- Variable auto-initialization.
- Transformations.

The following example illustrates the Visual Basic programming style:

```
Dim firstName As String
Dim lastName As String
Dim selectedButton As String

' Ask the person's name
Input "First Name:" : firstName,
      "Last Name:" : lastName, title := "Enter Your Name", buttons := { "Done",
"Cancel" }, Out selection := selectedButton

' Check the button pressed
If selectedButton = "Done" Then
    Display "Hello " & firstName & "!"
Else
    Display "Hello!"
End If
```

## Data Types

### Data Types Overview

Introduces PBL data types

PBL supports a number of data types, in four categories:

- [Numbers Overview](#) on page 256
- [String Overview](#) on page 266

- [Time and Interval Overview](#) on page 273
- [Booleans](#) Describes the Boolean data type

Often it is necessary to convert data types to other data types. For instance, a number can be converted to a string for display, or an input string may have to be converted to a date. For more details, see [Type Conversion](#) on page 256.

**Type Conversion**

Explains how to convert between data types

Conversion between variable types can be accomplished by "forcing" a type on a variable of another type. There are two syntaxes to make the conversion: *functional syntax* and the *conversion operator*.

**Functional Syntax**

The value to be converted is passed as an argument to the type name. Any variable type can be converted into a String. The following examples show you how to force a type on a variable:

```
someNum as Int = 23
someString as String

someString = String(someNum)
someString = String('now')
```

Any variable type can be converted from a String. However, this operation can fail if the format of the String is not valid for the type to which you are converting. For example, to convert to a Time data type, certain formats must be followed, as described in [Time and Interval Overview](#) on page 273.

```
localTime as Time
localTime = Time("2002/01/20 17:39:23")
```

The following example creates an Int from a String:

```
intNumber as Int
intNumber = Int("0001920")
```

**Conversion Operator**

Conversions can also be used by using the conversion operator to. The conversion operator is especially important when dealing with Transformations.

```
localTime as Time
localTime = "2002/01/20 17:39:23" to Time
```

**Numbers**

**Numbers Overview**

Describes numeric data types

Studio supports the following number data types:

Data Type	PBL Declaration
<a href="#">Integers</a> on page 257	Int
<a href="#">Reals</a> on page 257	Real
<a href="#">Decimals</a> on page 258	Decimal

Integers (type Int ) are generally used for counting and where whole numbers are suitable for the job. Decimals (type Decimal ) can be defined with a fixed decimal point and are particularly suited to store currency values.



Reals (type Real ) have a floating decimal point and can therefore adopt a very large range of values, but shouldn't be used for currency values due to rounding effects.

### Integers

Describes integer data types

Integers are whole numbers with no fractional part. In PBL all integer types are signed. Integers are suitable for storing unit quantities and are also used within program code as counters in loops or to handle various system values such as colors, character codes, and so on.

In PBL, integers constants can be specified in one of three bases:

Type	Example	Digits Allowed
octal	0567	0, 1, 2, 3, 4, 5, 6, 7
decimal	579	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
hexadecimal	0x5AF3	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

Octal (base 8) numbers are prefixed by a 0 (zero), and are rarely used. Hexadecimal (base 16) numbers are prefixed by '0x' and may be required for some types of values. However, in most situations with PBL, only decimal (base 10) integers will be used. These should not be confused with the Decimal data type, described below.

Integer variables have a fixed range of values that depends on their size in bits, or *precision*, as follows:

Precision	Maximum Value	Minimum Value
64	9,223,372,036,854,775,807	-9,223,372,036,854,775,808
32	2,147,483,647	-2,147,483,648
16	32,767	-32,768
8	127	-128

Precision is specified in parenthesis after the data type, as follows:

```
n As Int(<precision>)
```

For example, if you wish to declare a 32-bit Int variable, you would use:

```
n As Int(32)
```



**Note:** Only the four precision values shown in the table are defined for integers.

### Reals

Describes Real numbers

Real numbers are implemented as floating point numbers according to the IEEE 754 specification, which is the specification used by Java. All constants of type Real must include a period and they may have an exponential part denoted by 'e' or 'E'. They may also be denoted by the suffix 'f', 'F', 'd' or 'D'. The suffix is optional if the exponent is included.

Reals provide a fast and compact way of handling a very large range of values. However, they introduce rounding errors and are therefore not suitable when every digit, and particularly trailing digits, must be exact. This is a requirement when operating on monetary amounts, which should be stored as *Decimals* on page 258.

The following are all valid real constants:

- 2.0f
- 2.0E20
- 5.324d

Variables of type Real can be declared in one of two precisions, measured in bits of storage:

Precision	Maximum Value	Minimum Value
64	$1.7976931348623157 \times 10^{308}$	$4.9 \times 10^{-324}$
32	$3.4028235 \times 10^{38}$	$1.4 \times 10^{-45}$

The precision is specified in parenthesis after the data type, as follows:

```
x As Real(<precision>)
```

For example, if you wish to declare a 64-bit real, you would use:

```
x As Real(64)
```



**Note:** Only the two precision values shown in the table are supported.

## Decimals

Describes Decimal numbers

Decimal numbers are arbitrary precision numbers which do not use an exponent multiplier as reals may. Unlike reals, decimals must include every single digit. This eliminates the rounding problems of Reals but increases storage requirements for large values. As a general rule, decimals are used for currency amounts.

Each decimal value has a *precision* and a *scale*. The scale is the number digits to the right of the decimal separator. The precision is the total number of significant digits.

When you declare a decimal number you can do the following:

- Fix neither the precision nor the scale
- Fix only the scale
- Fix both scale and precision

The following example shows various ways of declaring decimal values:

```
unspecified as Decimal
fixedScale as Decimal(2)
fixedBoth as Decimal(5, 2)
```

In this example, `unspecified` is of arbitrary precision and scale, `fixedScale` can be any number with only two digits to the right of the decimal digit (e.g. 600000.25), and `fixedBoth` can hold only numbers with up to two digits to the right of the decimal point for a total of 5 digits.

Decimal constants are a sequence of decimal digits followed by a period (.), followed by another sequence of decimal digits. Note that unlike Real constants, the number of digits to the right of the decimal point is significant and specifies the scale of the constant. The scale affects how a number is displayed as in the following example:

```
display 12.3456780 to Decimal(2)
```

This example displays 12.35. Note that it *rounds*, and does not truncate, the original value.

## Decimal Arithmetic

It is very important to bear in mind the rules that apply to decimal arithmetic when dealing with variables with different decimal precisions. They are the following:

- If you want a variable to handle a determined precision, you must declare it: `Decimal(precision)`.
- In an addition or a subtraction, the result takes the highest precision of the two operands.
- In a multiplication, the precision of the result is the sum of the precisions of the two operands.
- In a division, the result has the precision of the left operand (the *dividend*).
- In an assignment:
  - it takes the precision resulting from the operation if the left operand has no precision defined or if its precision is higher.

- it takes the left operand precision when it is lower than the precision resulting from the operation.
- Every time the precision is reduced, the resulting value is *rounded*. For example, *0.5* is rounded to *1* and not to *0*.

### Assignment

```
dec4 = 10.20
dec2 = dec4
dec1 = dec4
dec3 = dec4
```

```
display 'dec1: ' + dec1 + ', dec2: ' + dec2 + ', dec3: ' + dec3 +
', dec4: ' + dec4
```

This example displays the following:

```
dec1:10.2000, dec2: 10.20, dec3: 10, dec4: 10.2
```

### Addition

```
dec1 = 10.20
dec2 = 1.04
dec3 = 100.003
```

```
res1 = dec1 + dec2
res2 = dec1 - dec2
res3 = dec1 + dec2 + dec3
res4 = dec3 - dec2
```

```
display 'res1: ' + res1 + ', res2: ' + res2 +
', res3: ' + res3 + ', res4: ' + res4
```

This example outputs the following:

```
res1:11.2400, res2:9.1600, res3: 111.2400, res4:98.96
```

### Multiplication

```
dec1 = 10.20
dec2 = 1.04
dec3 = 100.003
```

```
res1 = dec1 * dec2
res2 = dec2 * dec1
res3 = dec3 * dec1
res4 = dec1 * dec1
```

```
display 'res1: ' + res1 + ', res2: ' + res2 +
', res3: ' + res3 + ', res4: ' +
res4
```

This example outputs the following:

```
res1: 10.608000, res2:10.608000, res3: 1020.0000, res4: 104.04000000
```

### Division

```
dec1 = 10.20
dec2 = 1.04
dec3 = 100.003
res1 = dec1 / dec2
res2 = dec2 / dec1
res3 = dec3 / dec1
```

```
res4 = dec1 / dec1
display 'res1: ' + res1 + ', res2: ' + res2 +
', res3: ' + res3 + ', res4: ' +
res4
```

This example outputs the following:

```
res1: 9.8077, res2: 0.10, res3: 10, res4: 1.0000
```

## Real and Decimal Numbers

The choice between Real or Decimal numbers is important. Both can hold large numbers with a certain amount of precision, but there are fundamental differences between the two types of numbers:

- Real numbers are designed for speed in calculations where accuracy is not so important and where a close value is good enough.
- Decimal numbers are designed to provide a bound to the error you are willing to accept, sacrificing some performance if it is needed to guarantee the accuracy.

These differences become especially important when dealing with money. As a rule, whenever you are handling numbers that represent money, use Decimal numbers.

## Enumerations

Enumerations are sets of related integer constants, where each value has a name. Studio has built-in support for enumerations (sequential and non-sequential). These are some properties of enumerations:

- Type safe: The compiler checks that the values belong to the enumeration.
- User-Friendly: Understanding a piece of code that uses them is easier. Enumerations have a name that indicates what they represent.
- Improved Performance: Comparison of enumeration values is reduced to a comparison between integers.

## Using Enumerations

Enumerations can be used with a qualified name:

```
action = Action.SKI
```

or without qualification, when the type of the enumeration can be inferred from the context:

```
action = SKIP
```

To check the value of an Enumeration, you can use the is operator:

```
if action is CANCEL then
//Do something
end
```

or if you prefer, the multi-path conditional statement:

```
case action
when SKIP then
// Process SKIP
when CANCEL then
// Process CANCEL
else
// Handle all other values
end
```

## Creating Enumerations

For further information on creating enumerations, refer to: Enumerations

## Number Functions Reference

### abs

Returns the absolute value of a numeric value. If the argument is not negative, the argument is returned. If the argument is negative, the negation of the argument is returned.

Arguments	<b>Number</b>  numeric argument whose absolute value is to be determined
Returns	Numeric value of the same datatype as the input argument

The following special cases apply to this function:

- If the argument is positive, zero, or negative zero, the result is positive zero.
- If the argument is infinite, the result is positive infinity.
- If the argument is not a number (NaN), the result is also not a number.

### acos

Returns the arc cosine of an angle, in the range of 0.0 through pi.

Arguments	<b>Number</b>  Numeric argument whose arc cosine is to be determined. This value can range from -1 to 1.
Returns	Real - the arc tangent of the argument.

The following special cases apply to this function:

- If the argument is NaN or its absolute value is greater than 1, the result is NaN.

### asin

Returns the arc sine of an angle, in the range of -pi/2 through pi/2.

Arguments	<b>Number</b>  Numeric argument whose arc sine is to be determined. This value can range from -1 to 1.
Returns	Real - the arc sine of the argument.

The following special cases apply to this function:

- If the argument is NaN or its absolute value is greater than 1, the result is NaN.
- If the argument is zero, the result is a zero with the same sign as the argument.

### atan

Returns the arc tangent of an angle, in the range of -pi/2 through pi/2.

Arguments	<b>Number</b>
-----------	---------------

	Numeric argument whose arc sine is to be determined.
Returns	Real - the arc tangent of the argument.

The following special cases apply to this function:

- If the argument is NaN, the result is NaN.
- If the argument is zero, the result is a zero with the same sign as the argument.

### ceil

Returns the smallest (closest to negative infinity) real value which is not less than the argument and is equal to a mathematical integer.

Arguments	Real
Returns	Real - the smallest (closest to negative infinity) floating-point value that is not less than the argument and is equal to a mathematical integer.

The following special cases apply to this function:

- If the argument value is already equal to a mathematical integer, the result is the same as the argument.
- If the argument is positive zero or negative zero, the result is the same as the argument.
- If the argument value is less than zero but greater than -1.0, the result is negative zero.



**Note:** The value of `ceil(<Real>)` is exactly the value of `-floor(-<Real>)`.

### cos

Returns the trigonometric cosine of an angle.

Arguments	Real - an angle, in radians.
Returns	Real - the cosine of the argument.

The following special cases apply to this function:

- If the argument is NaN or an infinity, the result is NaN.

### exp

Returns Euler's number e raised to the power of a real value.

Arguments	Real - the exponent to raise e to.
Returns	Real - the value $e^a$ , where e is the base of the natural logarithms.

The following special cases apply to this function:

- If the argument is NaN, the result is NaN.
- If the argument is positive infinity, the result is positive infinity.
- If the argument is negative infinity, the result is positive zero.

### floor

Returns the largest (closest to positive infinity) real value that is not greater than the argument and is equal to a mathematical integer.

Syntax	<code>Real = floor (Real)</code>
Arguments	Real - a value.
Returns	Real - the largest (closest to positive infinity) floating-point value that is not greater than the argument and is equal to a mathematical integer.

The following special cases apply to this function:

- If the argument value is already equal to a mathematical integer, the result is the same as the argument.
- If the argument is NaN, an infinity, positive zero, or negative zero, the result is the same as the argument.

## log

Returns the natural logarithm (base e) of a numeric value.

Arguments	Number - a number greater than 0.
Returns	Real - the value $\ln$ argument, the natural logarithm of argument.

The following special cases apply to this function:

- If the argument is NaN or less than zero, the result is NaN.
- If the argument is positive infinity, the result is positive infinity.
- If the argument is positive zero or negative zero, the result is negative infinity.

## max(numA, numB)

Returns the greater of two numeric values. That is, the result is the argument closer to positive infinity. If the arguments have the same value, the result is that same value. If either value is NaN, the result is NaN. Unlike the numerical comparison operators, this method considers negative zero to be strictly smaller than positive zero. If one argument is positive zero and the other is negative zero, the result is positive zero (see the References in the footnotes).

Arguments	<b>numA</b>	numeric value A
	<b>numB</b>	numeric value B
Returns	Numeric value - the larger of the two arguments.	

## min(numA, numB)

Returns the smaller of two numeric values. That is, the result is the argument closer to the value of `Real.MIN_VALUE`. If the arguments have the same value, the result is that same value.

Arguments	<b>numA</b>	numeric value A
	<b>numB</b>	numeric value B
Returns	Numeric value - the smaller of the two arguments.	

**pow**

Returns the value of the first argument raised to the power of the second argument.

Arguments	Real - the base. Real - the exponent.
Returns	Real

The following special cases apply to this function:

- If the second argument is positive or negative zero, the result is 1.0.
- If the second argument is 1.0, the result is the same as the first argument.
- If the second argument is NaN, the result is NaN.
- If the first argument is NaN and the second argument is nonzero, the result is NaN.
- If
  - the absolute value of the first argument is greater than 1 and the second argument is positive infinity, or
  - the absolute value of the first argument is less than 1 and the second argument is negative infinity, the result is positive infinity.
- If
  - the absolute value of the first argument is greater than 1 and the second argument is negative infinity, or
  - the absolute value of the first argument is less than 1 and the second argument is positive infinity, the result is positive zero.
- If the absolute value of the first argument equals 1 and the second argument is infinite, the result is NaN.
- If
  - the first argument is positive zero and the second argument is greater than zero, or
  - the first argument is positive infinity and the second argument is less than zero, the result is positive zero.
- If
  - the first argument is positive zero and the second argument is less than zero, or
  - the first argument is positive infinity and the second argument is greater than zero, the result is positive infinity.
- If
  - the first argument is negative zero and the second argument is greater than zero but not a finite odd integer, or
  - the first argument is negative infinity and the second argument is less than zero but not a finite odd integer, the result is positive zero.
- If
  - the first argument is negative zero and the second argument is a positive finite odd integer, or
  - the first argument is negative infinity and the second argument is a negative finite odd integer, the result is negative zero.
- If
  - the first argument is negative zero and the second argument is less than zero but not a finite odd integer, or



- the first argument is negative infinity and the second argument is greater than zero but not a finite odd integer, the result is positive infinity.
- If
  - the first argument is negative zero and the second argument is a negative finite odd integer, or
  - the first argument is negative infinity and the second argument is a positive finite odd integer, the result is negative infinity.
- If the first argument is finite and less than zero
  - if the second argument is a finite even integer, the result is equal to the result of raising the absolute value of the first argument to the power of the second argument.
  - if the second argument is a finite odd integer, the result is equal to the negative of the result of raising the absolute value of the first argument to the power of the second argument
  - if the second argument is finite and not an integer, the result is NaN.
- If both arguments are integers, the result is exactly equal to the mathematical result of raising the first argument to the power of the second argument if that result can, in fact, be represented exactly as a real value.

In the foregoing descriptions, a floating-point value is considered to be an integer if and only if it is finite and a fixed point of the method `ceil` or, equivalently, a fixed point of the method `floor`. A value is a fixed point of a one-argument method if and only if the result of applying the method to the value is equal to the value.

### round

Returns the closest int to the argument. The result is rounded to a real by adding 1/2, taking the floor of the result and casting the result to type `int`. That is, `round` is equivalent to `floor(num + 0.5)`.

Arguments	<b>Number</b> numeric value to be rounded
Returns	Real or Decimal value of the argument rounded to the nearest whole number. If <code>num</code> is Real or Int, the function returns a Real. If <code>num</code> is Decimal, a Decimal is returned

The following special cases apply to this function:

- If the argument is NaN, the result is 0.
- If the argument is negative infinity or any value less than or equal to the value of `Real.MIN_VALUE`, the result is equal to the value of `Real.MIN_VALUE`.
- If the argument is positive infinity or any value greater than or equal to the value of `Real.MAX_VALUE`, the result is equal to the value of `Real.MAX_VALUE`.

### sin

Returns the trigonometric sine of an angle.

Arguments	Real - an angle, in radians.
Returns	Real - the sine of the argument.

The following special cases apply to this function:

- If the argument is NaN or an infinity, the result is NaN.
- If the argument is zero, the result is a zero with the same sign as the argument.

**sqrt**

Returns the correctly rounded positive square root of a Real value.

Arguments	Real - a value.
Returns	Real - the positive square root of argument. If the argument is NaN or less than zero, the result is NaN.

The following special cases apply to this function:

- If the argument is NaN or less than zero, the result is NaN.
- If the argument is positive infinity, the result is positive infinity.
- If the argument is positive zero or negative zero, the result is the same as the argument.
- Otherwise, the result is the real value closest to the true mathematical square root of the argument value.

**tan**

Returns the trigonometric tangent of an angle

Arguments	Real - an angle, in radians.
Returns	Real - the tangent of the argument.

The following special cases apply to this function:

- If the argument is NaN or an infinity, the result is NaN.
- If the argument is zero, the result is a zero with the same sign as the argument.

**Strings**  
**String Overview**

A string is zero or more characters put together. A character is anything that you can type, such as a letter, a digit, a symbol, or a space. Strings are used to hold any kind of text information.

**String literals**

A string literal consists of zero or more characters enclosed in double quotes. Each character may be represented by an escape sequence. The following are examples of string literals:

```
// empty string
display ""

// a string containing 16 characters
display "This is a string."

// a string containing one double quote
display "\""
```

In PBL style, consecutive strings are automatically merged:

```
display "This is a string "
      "made from separate strings."
```

The above code displays:

This is a string made from separate strings.

**Escape sequences**

Character and string escape sequences are used to denote some special characters as well as the single quote, double quote, and backslash characters in string literals. The following table lists the most common characters:

Escape code	Unicode	Description
\t	0009	Horizontal tab (HT)
\n	000a	Newline or line feed (LF)
\f	000c	Form feed (FF)
\r	000d	Carriage return (CR)
\'	0027	Single quote
\"	0022	Double quote
\\	005c	Backslash

If you know the Unicode code, any character can be specified. You must prefix the four-digit hexadecimal code of the desired character with "\u". For example, the double quote would be expressed as "\u0022".

### String concatenation

Strings can be concatenated at runtime with the string concatenation operator '+'. You can use any data type to build a string, so long as at least one of the elements is a string:

```
total as Int
total = 200
display "Total is: " + total + " units"
```

### Regular Expressions

Advanced string pattern matching and manipulation can be done with *regular expressions*. For further information, please see [Regular Expression Overview](#) on page 330.

### String Functions

#### substring

Returns a new string that is a substring of this string. The substring begins with the character at the specified index and extends to the end of this string.

#### Arguments

- String: the string on which the function operates.
- Int first: the beginning index, inclusive.

#### Returns

- String: the specified substring.

#### Example

```
text as String
text = "Hello World"
display substring(text, first : 5)
```

The previous example displays "World!".

#### substring

Returns a new string that is a substring of this string. The substring begins at the specified first index and extends to the character at index last - 1. Thus, the length of the substring is last - first.

#### Arguments

- String: the string on which the function operates.
- Int first: the beginning index, inclusive.
- Int last: the ending index, exclusive.

**Returns**

- String: the specified substring.

**Example**

```
text as String
text = "Hello World!"
display substring(text, first : 5, last : 11)
```

The previous example displays "World".

**fields**

Given a source string and delimiter character, it returns an array of strings containing substrings of the original string delimited by the specified character. The delimiter is not included in the result.

**Arguments**

- String: the string on which the function operates.
- String delim: Character that delimits field.

**Returns**

- String[]: an array of strings containing the fields delimited by **delim**.

**Example**

```
text as String
text = "Hello World!"
display fields(text, delim : " ")
```

The example above displays ["Hello", "World!"].

**length**

Returns the number of characters in the string.

**Arguments**

- String: the string on which the function operates.

**Returns**

- Int: Number of characters in the string.

**Example**

```
text as String
text = "Hello World!"
display length(text)
```

The previous example displays 12.

**replace**

Returns a new string with all the occurrences of **from** in the original string replaced by **to**.

This function has a variation that accepts a regular expression for matching the pattern to be replaced. See Regular Expressions for details.

**Arguments**

- String: the string on which the function operates.

- String from: the string to find.
- String to: the replacement text.

### Returns

- String: a new string with the replacements.

### Example

```
text as String
text = "Hello World!"
display replace(text, from : "World", @to : "Mary")
```

The previous example displays "Hello Mary!".

### charAt

This function returns the character contained in the specified index position.

### Arguments

- String: the string on which the function operates.
- Int position: zero-based index of a character inside the string.

### Returns

- String(1): the character at the specified index.

### Example

```
text as String
text = "Hello World!"
display charAt(text, position : 6)
```

The previous example displays "W".

### indexOf

Searches inside a string for another string and returns the index where the first occurrence happens.

This function has a variation that accepts a regular expression for matching. See Regular Expressions for details.

### Arguments

- String: the string on which the function operates.
- String text: the text to find.

### Returns

- Int: the index of the occurrence or -1 if not found.

### Example

```
text as String
text = "Hello World!"
display indexOf(text, text : "Wor")
```

The previous example displays 6.

### lastIndexOf

Searches inside a string for another string and returns the index where the last occurrence happens.

This function has a variation that accepts a regular expression for matching. See Regular Expressions for details.

### Arguments

- String: the string on which the function operates.
- String text: the text to find.

### Returns

- Int: the index of the occurrence or -1 if not found.

### Example

```
text as String
text = "Hello World!"
display lastIndexOf(text, text : "o")
```

The previous example displays 7.

### split

Splits a string using a regular expression. The delimiters are not included. See Regular Expressions for details.

### Example

```
text as String= "One Two Three"
display split(text, '/\w+ \w+/m')
```

The previous example produces this output [ "", "Three" ].

### count

This function counts the number of times that a character is found in a string.

### Arguments

- String: the string on which the function operates.
- String(1) ch: character to find.

### Returns

- Int: the number of occurrences of the specified character in the string.

### Example

```
date as String = "10/12/2004"
if count (date, ch: "/") = 2 then
    date = replace(date, "/", "-")
    display date
else
    display "Bad Date Format"
end
```

The previous example displays "10-12-2004".

### toUpperCase

Returns a new string with all the characters in uppercase.

### Arguments

- String: the string on which the function operates.

**Returns**

- String: a new string with all the characters in uppercase.

**Example**

```
text as String
text = "Hello World!"
display toUpperCase(text)
```

The previous example displays "HELLO WORLD!".

**toLowerCase**

Returns a new string with all the characters in lowercase.

**Arguments**

- String: the string on which the function operates.

**Returns**

- String: a new string with all the characters in lowercase.

**Example**

```
text as String
text = "Hello World!"
display toLowerCase(text)
```

The previous example displays "hello world!".

**trim**

Returns a new string with all the whitespace removed from the beginning and the end of the string.

**Arguments**

- String: the string on which the function operates.

**Returns**

- String: a new string with all the leading and trailing whitespace removed.

**Example**

```
option as String = " Yes  "
if toLowerCase(trim (option))= "yes" then
  display "The option is correct"
else
  display "The option is wrong".
end
```

**isMatch**

Checks whether a string matches a regular expression. See [Regular Expression Overview](#) on page 330 for details.

**Example**

```
text as String= "One Two Three"
display isMatch(text, '/\w+ Two \w+/'g')
```

The previous example displays true.

### **contains**

This function returns true if a substring in a text matches the specified regular expression. See Regular Expressions for details.

### **Example**

```
text as String= "One Two Three Four Five"
display contains(text, '/\w+ Tw/g')
```

The previous example displays true.

### **chars**

Returns an array of String(1) containing all the characters in the string.

### **Arguments**

- String: the string on which the function operates.

### **Returns**

- String(1[]): the characters in the string.

### **Example**

```
text as String= "fuego"
characters as String(1)[]
characters = chars(text)
for each i in characters
do
    display "Char is " + i
end
```

The previous example displays:

```
"Char is f"
"Char is u"
"Char is e"
"Char is g"
"Char is o"
```

### **pad**

Returns a new string completed with spaces until the specified length is reached.

### **Arguments**

- String: the string on which the function operates.
- Int len: the desired length of the string. If you pass a negative number (e.g.: -1), it is ignored and returns an empty string.

### **Returns**

- String: a new string of the specified length.

### **Example**

```
text as String
```



```
text = "Hello World!"
display pad(text, len : 20)
```

## strip

The `strip` function returns a new string truncated to a specified length. If the string is shorter than the length specified, it is left as it is.

### Arguments

- String: the string on which the function operates.
- Int len: the desired length of the string. If you pass a negative number (e.g.: -1), it is ignored and returns an empty string.

### Returns

- String: a new string of the specified length.

### Example

```
text as String
text = "Hello World!"
display strip(text, len : 5)
```

The previous example displays "Hello".

## How to convert a String to a Time

The following will convert the string in `strDate` into a `Time` value to be set in `realDate`.

```
strDate = "Tue Feb 22 15:26:02 ART 2005"
strPattern = "EEE MMM dd HH:mm:ss z yyyy"

simpleDateFormat = Java.Text.SimpleDateFormat(strPattern)
realDate = parse(simpleDateFormat, strDate)

display realDate
```

The value of `realDate` becomes **2005-02-22 15:26:02-03**

`realDate` is a variable of type `Time`. It contains the same date that was expressed as a string in `strDate`.

## String Attributes

### Empty

This attribute returns true if the string is empty (its length is zero).

### Returns

- Boolean: true if its length is zero. False otherwise.

### Example

```
text as String
text = "Hello World!"
display text.empty
```

## Times and Intervals

### Time and Interval Overview

Studio supports two built-in types for time management:

- Time - represents a specific point in time.
- Interval - represents the difference between two moments in time.

Times are stored as the number of microseconds since Jan 1, 1970 (also known as UNIX epoch).

Intervals are stored as months, days, and microseconds.

## Times

The string representation of a time is ISO 8601 compliant. When converting a string into a time, Studio supports a somewhat relaxed subset of ISO 8601; even dates without separators are accepted. The accepted string formats match those of time literals.

## Time Literals

Time literals are specified between single quotes. The following is a list of valid time literals:

```
'23:30'
'23:30:23'
'23:30:23.001023'
'23:30:23.001023Z'
'23:30:23.001023-05'
'23:30:23.001023-3:30'
'1995-02-03'
'1995-02-03 23:30'
'1995-02-03 23:30:23'
'1995-02-03 23:30:23.001023'
'1995-02-03 23:30:23.001023Z'
'1995-02-03 23:30:23.001023-05'
'1995-02-03 23:30:23.001023-3:30'
'1995-02-03T23:30'
'1995-02-03T23:30:23'
'1995-02-03T23:30:23.001023'
'1995-02-03T23:30:23.001023Z'
'1995-02-03T23:30:23.001023-05'
'1995-02-03T23:30:23.001023-3:30'
'19950203T'
'19950203T233023.001023-330'
```

## Time Zones

Following ISO 8601, time zones are specified as offsets from UTC. Named time zones are not supported because there is no international standard for time zone abbreviations. If no time zone is specified in a time literal, the default time zone for the current locale is used.

Note that:

- Interactive methods run in the locale of the current user.
- Automatic methods run in the Process Execution Engine's locale.

When a time is presented to a user, the format associated to the user's locale is used. For custom time formatting, the format function can be used. For further information, please see Time Functions.

## Intervals

Intervals have two primary parts:

- Two calendar dependent components (months and days)
- A calendar independent component (hours, minutes, seconds and microseconds)

The calendar dependent component exists, so arithmetic between time and interval is consistent. When using a Gregorian calendar (the most common calendar in use), you cannot assume that a month equals to 30 days. In fact, you cannot even assume that a day lasts 24 hours.

The Gregorian calendar inserts two corrections:

- The leap year - Every four years a day is added to February, unless the year is a multiple of 100 and not a multiple of 400 (which is why the year 2000 had 366 days, instead of 365 like 1900).
- The leap second - Sometimes a second is added or removed from the last minute of certain days to cope with the accumulated error caused by the Earth's change of speed.

So, for example, if you want to obtain a time two months from now, you can do:

```
display 'now' + '2M'
```

### Interval Literals

Interval literals are enclosed by single-quotes ('). They are formed by a sequence of fields, where each field is a number plus a unit suffix. The following table lists all valid suffixes:

Unit Suffix	Description
Y	Years
M	Months
d or D	Days
h or H	Hours
m	Minutes
s or S	Seconds
x	Microseconds



**Note:** A 'T' in the constant forces the interpretation of 'M' to be equal to 'm', e.g.: '2MT2M' equals '2M2m'.

All magnitudes can contain a '.' to express a fractional part, although the fractional part is dropped for days or months.

The following example shows some valid Interval constants:

```
display '2MT2.5M'
display '1Y1M3h2m1.500s'
display '1.5h'
```

### Arithmetic

As mentioned before, time and intervals have some arithmetic rules.

The following table lists the behavior of addition and subtraction with time and interval:

Operations	Result Type
Time - Time	Interval
Time + Interval (or Interval + Time)	Time
Time - Interval	Time
Interval + Interval	Interval
Interval - Interval	Interval

For further information on Time and Interval, please refer to the following:

- Time Attributes
- Interval Attributes
- Time and Interval Functions

**Time Attributes**

A Time object contains several attributes to manipulate the different components of time, such as days and hours. The following table lists all Time's attributes and provides some examples:

Attribute	Description
AD	Field that indicates the calendar era indicating the common era (Anno Domini.).

Example:

```
time as Time
/*this will display 1 because
the current era is AD*/
display time + "\n\n AD = " + time.AD
```

Attribute	Description
am	Boolean value which indicates whether the Time's period is between midnight to just before noon.

Example:

```
time as Time
time = '2004-12-25 02:45:00-03'
//this will display true
display time + "\n\n am = " + time.am
time = '2004-12-25 20:45:00-03'
display time + "\n\n am = " + time.am
```

Attribute	Description
ampm	Field that indicates the period of the day. If the period is am, it will return 0; otherwise, it will return 1.

Example:

```
time as Time
time = '2004-12-25 02:45:00-03'
//this will display 0
display time + "\n\n ampm = " + time.ampm
time = '2004-12-25 20:45:00-03'
//this will display 1
display time + "\n\n ampm = " + time.ampm
```

Attribute	Description
BC	Field that indicates the calendar era indicating the period before the common era (before Christ).

Example:

```
time as Time
/*this will display 0 because
the current era is not BC*/
display time + "\n\n BC = " + time.BC
```

Attribute	Description
date	String value containing the date formatted with the default mask.

Example:

```
time as Time
time = '2004-12-25 20:45:00-03'
//this will display Dec 25, 2004
display time + "\n\n" + time.date
```

Attribute	Description
day	Int value representing the day component of the time.

Example:

```
time as Time
time = '2004-12-25 20:45:00-03'
//it will display 25
display time + "\n\n day: " + time.day
```

Attribute	Description
dayOfMonth	Int value representing the day part of the time.

Example:

```
time as Time
time = '2004-12-25 20:45:00-03'
//this will display 25
display time + "\n\n day of month: " +
time.day
```

Attribute	Description
dayOfWeek	Day of the week. Returns an integer from 1 to 7, where Sunday is 1.

Example:

```
time as Time
time = '2004-12-25 20:45:00-03'
display time + "\n\n day of the week: " +
time.dayOfWeek
```

Attribute	Description
days	Returns the number of days elapsed since 00:00 January 1, 1970 GMT (UNIX epoch).

Example:

```
time as Time
display "days since EPOCH: " + time.days
```

Attribute	Description
EPOCH	1969-12-31 00:00:00-00. Base time from which milliseconds to calculate dates are counted.

Example:

```
//1969-12-31 00:00:00-00
display "EPOCH: " + Time.EPOCH
```

Attribute	Description
firstDayOfMonth	Time value corresponding to the first day of the month.

Example:

```
time as Time
//firstDayOfMonth is a Time object
display "fist day of month: " +
time.firstDayOfMonth
```

Attribute	Description
hour	Int value representing the hour component of the date in the format h.

Example:

```
time as Time
time = '2004-12-25 20:45:00-03'
display time + "\n\n hour: " + time.hour
```

Attribute	Description
hourOfDay	Int value representing the hour component of the date in the format hh.

Example:

```
time as Time
time = '2004-12-25 20:45:00-03'
display time + "\n\n hour of the day: " +
time.hour
```

Attribute	Description
hours	Returns the number of hours elapsed since January 1, 1970 GMT.

Example:

```
time as Time
display "hours passed since EPOCH: "+
time.hours
```

Attribute	Description
lastDayOfMonth	Time value corresponding to the last day of the month.

Example:

```
time as Time
//lastDayOfMonth is a Time object
display "last day of month: " +
time.lastDayOfMonth
```

Attribute	Description
locale	This attribute is used to change the current locale. It is a write only attribute.

Example:

```
time as Time
display "date fomatted with default locale: " +
time.formatDate
Time.locale = Java.Util.Locale.GERMAN
display "date with German locale: " +
time.formatDate
```

Attribute	Description
maxvalue	The maximum value a Time object can have.

Example:

```
display "Time object's maximum value: "
+ Time.maxvalue
```

Attribute	Description
microSecond	Int value representing the microseconds component of the date.

Example:

```
time as Time
display time + "\n\n microseconds in time: " +
time.microSecond
```

Attribute	Description
microSeconds	Returns the number of microseconds elapsed since January 1, 1970 GMT.

Example:

```
time as Time
display "microseconds since EPOCH: " +
time.microDeconds
```

Attribute	Description
milliSeconds	Returns the number of milliseconds elapsed since January 1, 1970 GMT.

Example:

```
time as Time
display "milliseconds since EPOCH: " +
time.milliSeconds
```

Attribute	Description
minute	Int value representing the minutes component of the date.

Example:

```
time as Time
time = '2004-12-25 20:45:00-03'
display time + "\n\n minutes: " + time.minute
```

Attribute	Description
minutes	Returns the number of minutes elapsed since January 1, 1970 GMT.

Example:

```
time as Time
display "minutes since EPOCH: " + time.minutes
```

Attribute	Description
minvalue	The minimum value a Time object can have.

Example:

```
time as Time
display "Time object's minimum value: "
+ time.minvalue
```

Attribute	Description
month	Int value representing the month component of the date.

Example:

```
time as Time
time = '2004-12-25 20:45:00-03'
display time + "\n\n month: " +
time.month
```

Attribute	Description
second	Int value representing the seconds component of the date.

Example:

```
time as Time
time = '2004-12-25 20:45:10-03'
display time + "\n\n seconds: " + time.second
```

Attribute	Description
seconds	Returns the number of seconds elapsed since January 1, 1970 GMT.



Example:

```
time as Time
display "seconds since EPOCH: " + time.seconds
```

Attribute	Description
timeOnlyHour	Int value representing the hours component of the time, without calendar corrections.

Example:

```
time as Time
display time + "\n\n" +
"hours: " +
time.timeOnlyHour
```

Attribute	Description
timeOnlyMicroSecond	Int value representing the microseconds component of the time, without calendar corrections.

Example:

```
time as Time
display time + "\n\n" +
"microseconds: " +
time.timeOnlyMicroSecond
```

Attribute	Description
timeOnlyMinute	Int value representing the minutes component of the time, without calendar corrections.

Example:

```
time as Time
display time + "\n\n"+
"minutes: " +
time.timeOnlyMinute
```

Attribute	Description
timeOnlySecond	Int value representing the seconds component of the time, without calendar corrections.

Example:

```
time as Time
display time + "\n\n"+
"seconds: " +
time.timeOnlySecond
```

Attribute	Description
time	String value containing the time of the day formatted with the default mask.

Example:

```
time as Time
display "time formatted with default mask:" +
time.date
```

Attribute	Description
timeZone	Time according to the locale.

Example:

```
time as Time
Time.timeZone = TimeZone.getTimeZone( "GMT-3" )
display "GMT-3: " + time
Time.timeZone = TimeZone.getTimeZone( "GMT-8" )
display "GMT-8: " + time
```

Attribute	Description
weekOfMonth	Int value indicating the week number within the current month, starting from 1.

Example:

```
time as Time
time = '2004-01-01 20:45:00-03'
display time + "\n\n week of month: " +
time.weekOfMonth
time = '2004-01-07 20:45:00-03'
display time + "\n\n week of month: " +
time.weekOfMonth
```

Attribute	Description
weekOfYear	Int value indicating the week number within the current year, starting from 1.

Example:

```
time as Time
time = '2004-01-07 20:45:00-03'
display time + "\n\n week of year: " +
time.weekOfYear
time = '2004-02-07 20:45:00-03'
display time + "\n\n week of year: " +
time.weekOfYear
```

## Time Functions

### addDays

Adds a specified number of days to a Time object.

#### Arguments

- Time - the Time object to which days will be added.
- Int i - the number of days to be added to the time object.

#### Returns

The Time object resulting from adding the specified number of days to the given time.

### Example

The following example adds 15 days to the current time and displays the result.

```
display "time in 15 days will be: " +
```

```
addDays('now', i : 15)
```

### **addHours**

Adds a specified number of hours to a Time object.

#### **Arguments**

- Time - the Time object to which hours will be added.
- Int i - the number of hours to be added to the Time object.

#### **Returns**

The Time object resulting from adding the specified number of hours to the given time.

#### **Example**

The following example adds 12 hours to the current time and displays the result:

```
display "time in 12 hours will be: " +
    addHours('now', i : 12)
```

### **addMicroSeconds**

Adds a specified number of microseconds to a Time object.

#### **Arguments**

- Time - the Time object to which microseconds will be added.
- Int i - the number of microseconds to be added to the Time object.

#### **Returns**

The Time object resulting from adding the specified number of microseconds to the given time.

#### **Example**

The following example adds 500 microseconds to the current time and displays the result:

```
display "time in 500 microseconds will be: " +
    addMicroSeconds('now', i : 500)
```

### **addMilliSeconds**

Adds a specified number of milliseconds to a Time object.

#### **Arguments**

- Time - the Time object to which milliseconds will be added.
- Int i - the number of milliseconds to be added to the Time object.

#### **Returns**

The Time object resulting from adding the specified number of milliseconds to the given time.

#### **Example**

The following example adds 50,000 milliseconds to the current time and displays the result:

```
display "time in 50000 milliseconds will be: " +
    addMilliSeconds('now', i : 50000)
```

### **addMinutes**

Adds a specified number of minutes to a Time object.

#### **Arguments**

- Time - the Time object to which the milliseconds will be added.
- Int i - the number of minutes to be added to the Time object.

**Returns**

The Time object resulting from adding the specified number of milliseconds to the given time.

**Example**

The following example adds 30 minutes to the current time and displays the result:

```
display "time in 30 minutes will be: " +
    addMinutes('now', i : 30)
```

**addMonths**

Adds a specified number of months to a Time object.

**Arguments**

- Time - the Time object to which the months will be added.
- Int i - the number of months to be added to the Time object.

**Returns**

The Time object resulting from adding the specified number of months to the given time.

**Example**

The following example adds 6 months to the current time and displays the result.

```
display "time in 6 months will be: " +
    addMonths('now', i : 6)
```

**addSeconds**

Adds a specified number of seconds to a Time object.

**Arguments**

- Time - the Time object to which the seconds will be added.
- Int i - the number of seconds to be added to the Time object.

**Returns**

The Time object resulting from adding the specified number of seconds to the given time.

**Example**

The following example adds 50 seconds to the time object and displays the result:

```
display "time in 50 seconds will be: " +
    addSeconds('now', i : 50)
```

**addWeeks**

Adds a specified number of weeks to a Time object.

**Arguments**

- Time - the Time object to which the weeks will be added.
- Int i - the number of weeks to be added to the Time object.

**Returns**

The Time object resulting from adding the specified number of weeks to the given time.

**Example**

The following example adds 50 weeks to the current time and displays the result:

```
display "time in 50 weeks will be: " +
    addWeeks('now', weeks : 50)
```

**addYears**

Adds a specified number of years to a Time object.

#### Arguments

- Time - the Time object to which the years will be added.
- Int i - the number of years to be added to the Time object.

#### Returns

The Time object resulting from adding the specified number of years to the given time:

```
display "time in 10 years will be: " +
    addYears('now', i : 10)
```

#### add

Adds a specified interval of time to a Time object.

#### Arguments

- Time - the Time object to which the interval of time will be added.
- Interval i - interval of time to add to the Time object.

#### Returns

The Time object resulting from adding the specified interval to the given time.

#### Example

The following example adds 5 days, 15 hours, and 30 minutes to the current time and displays the result:

```
display "time in 5 days 15 hours and 30 minutes: \n\n" +
    add('now', interval : '5d15h30m')
```

#### between

Determines if the given Time object is between two Time objects.

#### Arguments:

- Time - the given Time object.
- Time from - the upper bound of the time period in which to search the given time, exclusive.
- Time to - the lower bound of the time period in which to search the given time, inclusive.

#### Returns

true - if the given Time object is contained in the specified period.

false - if the given object is not contained in the specified period.

#### Example

The following example finds out if the current time is in between the first day and last day of year 2004:

```
display "is today between 2004-01-01 and 2004-12-31? "+
    between('now', from : '2004-01-01 12:00:00',
    @to : '2004-12-31 12:00:00')
```

#### daysSince

Calculates the days passed between a given time and another time.

#### Arguments

- Time - the Time object.
- Time t - the other Time object.

#### Returns

An Int value representing the number of days between the two given times.

**Example**

The following example defines a Time variable birthdate, calculates the days passed between 'now' and birthdate and displays the result:

```
birthdate as Time
  birthdate = '1979-02-19'
  display "days passed since birthdate: " +
    daysSince('now', t : birthdate)
```

**formatDate**

Formats the Time object with the default mask.

**Arguments**

- Time - the Time object to be formatted.

**Returns**

A String containing the representation of the Time object formatted with the default mask.

**Example**

The following example displays the current time formatted with the default mask:

```
display formatDate('now')
```

**formatDate**

Formats the Time object with the specified date formatting style for the default locale.

**Arguments**

- Time - the Time object to be formatted.
- Int style - The formatting style. Available styles are Time.DEFAULT, Time.FULL, Time.LONG, and Time.SHORT.

**Returns**

A String containing the representation of the Time object formatted with the specified style.

**Example**

The following example displays the current time with the four possible formatting styles:

```
defaultDate as String = "Default format --> " +
  formatDate('now', dateStyle : Time.DEFAULT)

fullDate as String = "Full format --> " +
  formatDate('now', dateStyle : Time.FULL)

longDate as String = "Long format --> " +
  formatDate('now', dateStyle : Time.LONG)

shortDate as String = "Short format --> " +
  formatDate('now', dateStyle : Time.SHORT)

display defaultDate + "\n\n" + fullDate + "\n\n" +
  longDate + "\n\n" + shortDate + "\n\n"
```

**formatTimeOnly**

Formats this Time object as a time only, with no time zone correction.

**Arguments**

- Time - The Time object to be formatted.

**Returns**

A String containing the representation of the Time object formatted as time only.

The following example displays the current time formatted as time only. It should display something similar to 12439d 16:58:58:

```
display formatTimeOnly('now')
```

**formatTimeOnly**

Formats this Time object as a time only, with no time zone correction, applying the specified style.

**Arguments**

- Time - The Time object to be formatted.
- Int - The formatting style. Available styles are Time.DEFAULT, Time.FULL, Time.LONG, and Time.SHORT.

**Returns**

A String containing the representation of the Time object formatted as time only, applying the specified style.

**Example**

The following example only displays the time representation of the current time in the four available styles:

```
defaultTime as String = "Default format --> " +
    formatTimeOnly('now', intervalStyle : Time.DEFAULT)

fullTime as String = "Full format --> " +
    formatTimeOnly('now', intervalStyle : Time.FULL)

longTime as String = "Long format --> " +
    formatTimeOnly('now', intervalStyle : Time.LONG)

shortTime as String = "Short format --> " +
    formatTimeOnly('now', intervalStyle : Time.SHORT)

display defaultTime + "\n\n" +
    fullTime + "\n\n" +
    longTime + "\n\n" +
    shortTime + "\n\n"
```

**formatTime**

Formats the time component of a Time object with a specified style.

**Arguments**

- Time - The Time object to be formatted.
- Int timeStyle - The formatting style. Available styles are Time.DEFAULT, Time.FULL, Time.LONG, and Time.SHORT.

**Returns**

A String containing the time component of the Time object formatted with the specified style.

**Example**

The following example displays the time component of the current time formatted with the four available styles:

```
defaultTime as String = "Default format --> " +
    formatTime('now', timeStyle : Time.DEFAULT)
```

```

fullTime as String = "Full format --> " +
    formatTime('now', timeStyle : Time.FULL)

longTime as String = "Long format --> " +
    formatTime('now', timeStyle : Time.LONG)

shortDate as String = "Short format --> " +
    formatTime('now', timeStyle : Time.SHORT)

display defaultTime + "\n\n" +
    fullTime + "\n\n" +
    longTime + "\n\n" +
    shortTime + "\n\n"

```

**format**

Formats a Time object with the default mask.

**Arguments**

- Time - the Time object to be formatted.

**Returns**

A string containing the representation of the Time object formatted with the default mask.

**Example**

The following example displays the current time formatted with the default mask.

```
display format('now')
```

**format**

Formats a Time object with a specified date style and time style.

**Arguments**

- Time - The Time object to be formatted.
- Int dateStyle - The style to be applied to the date component of the Time object. Available styles are: Time.DEFAULT, Time.FULL, Time.LONG, and Time.SHORT.
- Int timeStyle - The style to be applied to the time component of the Time object. Available styles are Time.DEFAULT, Time.FULL, Time.LONG, and Time.SHORT.

**Returns**

A string containing the representation of the Time object whose date was formatted with the specified date style, and whose time was formatted with the specified time style.

**Example**

The following example displays the current time with its date in full format style and its time in short format style first, then the same time with its date in short format style and its time in full format style:

```

fullDateShortTime as String = "Full date, short time: " +
    format('now', dateStyle : Time.FULL,
        timeStyle : Time.SHORT)

shortDateFullTime as String = "Short date, full time: " +
    format('now', dateStyle : Time.SHORT,
        timeStyle : Time.FULL)

display fullDateShortTime + "\n\n" + shortDateFullTime

```



**format**

Formats a Time object by applying a specified formatter.

**Arguments**

- Time - The Time object to be formatted.
- Java.Text.DateFormat formatter - The formatter to be applied in order to format the Time object.

**Returns**

A String containing the representation of the Time object formatted by applying the specified formatter.

**Example**

The following example displays the current Time formatted with the formatter passed by arguments:

```
display format('now', formatter : DateFormat.getInstance())
```

**format**

Formats a Time object with a specified formatter using the provided time zone and locale.

**Arguments**

- Time - The Time object to be formatted.
- Java.Text.DateFormat formatter - The formatter to be applied in order to format the Time object.
- Java.Util.TimeZone timeZone - The time zone to apply to the Time object when formatting it.
- Java.Util.Locale locale - The locale to apply to the Time object when formatting it.

**Returns**

A String containing the representation of the Time object formatted by applying the specified formatter and the time zone and locale provided.

**Example**

The following example displays the current Time formatted applying the formatter passed by arguments, GMT-10 time zone and French locale:

```
display format('now', formatter : DateFormat.getInstance(),
    timeZone : TimeZone.getTimeZone(arg1 : "GMT-10"),
    locale : Java.Util.Locale.FRANCE)
```

**format**

Formats a Time object with a specified mask.

**Arguments**

- Time - The Time object to be formatted.
- String mask - A string containing the mask to apply in order to format the Time object.

**Returns**

A String containing the representation of the Time object formatted with the specified mask. The String should be written according to the patterns and rules described below.

The following pattern letters are defined (all other characters from 'A' to 'Z' and from 'a' to 'z' are reserved):

Letter	Date or Time Component	Presentation	Example
G	Era designator	Text	AD
y	Year	Year	1996; 96
M	Month in year	Month	July; Jul; 07

Letter	Date or Time Component	Presentation	Example
w	Week in year	Number	27
W	Week in month	Number	2
D	Day in year	Number	189
d	Day in month	Number	10
F	Day of week in month	Number	2
E	Day in week	Text	Tuesday; Tue
a	Am/pm marker	Text	PM
H	Hour in day (0-23)	Number	0
k	Hour in day (1-24)	Number	24
K	Hour in am/pm (0-11)	Number	0
h	Hour in am/pm (1-12)	Number	12
m	Minute in hour	Number	30
s	Second in minute	Number	55
S	Millisecond	Number	978
z	Time zone	General time zone	Pacific Standard Time; PST; GMT-08:00
Z	Time zone	RFC 822 time zone	-0800

Pattern letters are usually repeated, as their number determines the exact presentation:

- Text: For formatting, if the number of pattern letters is 4 or more, the full form is used. Otherwise, a short or abbreviated form is used if available. For parsing, both forms are accepted, independent of the number of pattern letters.
- Number: For formatting, the number of pattern letters is the minimum number of digits, and shorter numbers are zero-padded to this number. For parsing, the number of pattern letters is ignored unless it's needed to separate two adjacent fields.
- Year: For formatting, if the number of pattern letters is 2, the year is truncated to 2 digits. Otherwise, it is interpreted as a number.

### Example

The following example displays the current time formatted with the mask defined by the String passed by arguments. It should display something like: Thu 22 01 2004 04:31:48:975 PM ART:

```
display 'now'.format("E dd MM yyyy hh:mm:ss:SS a z")
```

### getDateFormat

Returns an appropriate DateFormat based on the parts needed and a style.

### Arguments

- Int parts - The needed parts. These could be Time.DATE\_ONLY, Time.TIME\_ONLY, or Time.DATE\_TIME.
- Int style - The formatting style. Available styles are Time.DEFAULT, Time.FULL, Time.LONG, and Time.SHORT.

### Returns

The format based on the required parts and style.

### Example

The following example uses the function `getDateFormat` with different needed parts and style to format the current time:

```
fullDateOnly as String
fullDateOnly = "Date only - Full format: " +
  'now'.format(Time.getDateFormat(
    parts : Time.DATE_ONLY,
    style : Time.FULL))

shortTimeOnly as String
shortTimeOnly = "Time only - Short format: " +
  'now'.format(Time.getDateFormat(
    parts : Time.TIME_ONLY,
    style : Time.SHORT))

longDateTime as String
longDateTime = "Date time - Long format: " +
  'now'.format(Time.getDateFormat(
    parts : Time.DATE_TIME,
    style : Time.FULL))

display fullDateOnly + "\n\n" + shortTimeOnly + "\n\n" +
  longDateTime
```

## **getDateFormat**

Returns an appropriate `DateFormat` based on the required parts, style, and locale.

### **Arguments**

- `Int parts` - The needed parts. These could be `Time.DATE_ONLY`, `Time.TIME_ONLY`, or `Time.DATE_TIME`.
- `Int style` - The formatting style. Available styles are `Time.DEFAULT`, `Time.FULL`, `Time.LONG`, and `Time.SHORT`.
- `Java.Util.Locale` - The locale the date formatter will have to apply.

### **Returns**

Returns an appropriate `DateFormat` based on the required parts, style, and locale.

### **Example**

The following example uses the function `getDateFormat` with different needed parts and style, and french locale, to format the current time:

```
fullDateOnly as String
fullDateOnly = "Date only - Full format: " +
  'now'.format(Time.getDateFormat(
    parts : Time.DATE_ONLY,
    style : Time.FULL, Java.Util.Locale.FRANCE))

shortTimeOnly as String
shortTimeOnly = "Time only - Short format: " +
  'now'.format(Time.getDateFormat(
    parts : Time.TIME_ONLY,
    style : Time.SHORT, Java.Util.Locale.FRANCE))

longDateTime as String
longDateTime = "Date time - Long format: " +
  'now'.format(Time.getDateFormat(
    parts : Time.DATE_TIME,
    style : Time.FULL, Java.Util.Locale.FRANCE))
```

```
display fullDateOnly + "\n\n" + shortTimeOnly + "\n\n" +
      longDateTime
```

**Time.getEaster**

Calculates Easter day for a specified year.

**Arguments**

**y**

The year for which you would like to know Easter's date.

**Returns**

A Time object containing Easter date.

**Example**

The following example displays Easter date for the year 2007:

```
display Time.getEaster(y : 2007)
```

This will show:

```
Apr 8, 2007 12:00:00 AM
```

**Time.getMonthName**

Returns the name of the specified month.

**Arguments**

**monthNum**

The number which identifies the month from which you would like to know the name. This number may vary between 1 (January) and 12 (December).

**Returns**

The name of the specified month.

**Example**

The following example displays the name of month 2 (February):

```
display Time.getMonthName(monthNum : 2)
```

**Time.getWeekdayName**

Returns the name of the specified weekday.

**Arguments**

- **Int dayNum** - The number which identifies the day from which you would like to know the name. This number may vary between 0 (Sunday) and 6 (Saturday).

**Returns**

The name of the specified day.

**Example**

The following example displays the day 3 (Wednesday):

```
display Time.getWeekdayName(dayNum : 0)
```

**max**

Returns the greater time between two Time objects.

**Arguments****Time1**

A Time object

**Time2**

Another Time object

**Returns**

The greater Time object.

**Example**

The following example displays the greater Time between the current time and the date the man landed on the moon. The current time is returned:

```
manOnMoon as Time
manOnMoon = '1969-07-21 02:56:00 -00'
display max(manOnMoon, b : 'now')
```

**min**

Returns the smaller Time object between two Time objects.

**Arguments**

- Time a - A Time object.
- Time b - Another Time object.

**Returns**

The smaller Time object.

For example:

```
manOnMoon as Time
manOnMoon = '1969-07-21 02:56:00 -00'
display min(manOnMoon, b : 'now')
```

**monthsSince**

Returns the number of months elapsed since a given Time object.

**Arguments****Time**

A Time object

**Returns**

An Int value with the number of whole months elapsed since the time specified by the Time object to the present time. For example, there are two whole months between January 15 and March 15, but only one whole month between January 15 and March 14.

For example:

```
xMas2007 as Time
xMas2007 = '2007-12-25'
display monthsSince('now', t : xMas2007)
```

**roundToSeconds**

Rounds a Time object to the nearest second.


Arguments

Time

A Time object

Returns

A Time object with the value of Time rounded to the nearest second.

 **Note:** The value is rounded, not truncated, so 499 milliseconds or less will be rounded down, while 500 milliseconds or more will be rounded up.

Example

The following example creates a Time object called `timeStart` with seconds and milliseconds, then it displays the value rounded to the nearest second:

```
timeStart as Time
time = '2007-04-16 17:21:30.235'
display roundToSeconds(timeStart)
```

This displays:

```
Apr 16, 2007 2:21:30 PM
```

Interval Attributes

Attribute	Description
ONE_DAY	Interval value representing an interval of one day.

Example:

```
interval as Interval
interval = '25d5h1m'
display "original interval: " + interval
interval = interval + Interval.ONE_DAY
display "after adding a day: " + interval
```

Attribute	Description
ONE_HOUR	Interval value representing an interval of one hour.

Example:

```
interval as Interval
interval = '25d5h1m'
display "original interval: " + interval
interval = interval + Interval.ONE_HOUR
display "after adding an hour: " + interval
```

Attribute	Description
ONE_MINUTE	Interval value representing an interval of one minute.

Example:

```
interval as Interval
interval = '25d5h1m'
display "original interval: " + interval
interval = interval + Interval.ONE_MINUTE
display "after adding a minute: " + interval
```

Attribute	Description
ONE_MONTH	Interval value representing an interval of one month.

Example:

```
interval as Interval
interval = '2M25d5h1m'
display "original interval: " + interval
interval = interval + Interval.ONE_MONTH
display "after adding a month: " + interval
```

Attribute	Description
ONE_SECOND	Interval value representing an interval of one second.

Example:

```
interval as Interval
interval = '25d5h1m7s'
display "original interval: " + interval
interval = interval + Interval.ONE_SECOND
display "after adding a second: " + interval
```

Attribute	Description
ZERO	Interval representing the zero value. When added to another interval, the interval does not change its value.

Example:

```
interval as Interval
interval = '1M20d3h40m5s'
display "original interval: " + interval
interval = interval + Interval.ZERO
display "after adding ZERO: " + interval
```

Attribute	Description
daysOnly	Int value representing the days component of an interval.

Example:

```
interval as Interval
interval = '1M20d10h'
display "days component of an interval: " +
interval.daysOnly
```

Attribute	Description
days	Int value representing the days component of an interval.

Example:

```
interval as Interval
interval = '1M20d10h'
display "days component of an interval: " +
```

```
interval.days
```

Attribute	Description
hoursOnly	Int value representing the hours component of an interval.

Example:

```
interval as Interval
interval = '1d20h30m'
display "hours component of an interval: " +
interval.hoursOnly
```

Attribute	Description
hours	Int value representing the hours component of an interval.

Example:

```
interval as Interval
interval = '1d20h30m'
display "hours component of an interval: " +
interval.hours
```

Attribute	Description
microSecondsOnly	Int value representing the microseconds component of an interval.

Example:

```
interval as Interval
interval = Interval("20.000003s")
display "microseconds component of an interval: "+
interval.microSecondsOnly
```

Attribute	Description
microSeconds	Total number of microseconds contained in the interval.

Example:

```
interval as Interval
interval = Interval("20.000003s")
display "total microseconds of an interval: " +
interval.microSeconds
```

Attribute	Description
milliSecondsOnly	Int value representing the milliseconds component of an interval, without including microseconds.

Example:

```
interval as Interval
interval = '20.250320s'
```



```
display "milliseconds component of an interval: " +
interval.milliSecondsOnly
```

Attribute	Description
milliSeconds	Int value representing the milliseconds component of an interval plus the microseconds contained in it.

Example:

```
interval = '20.250320s'
display "milliseconds component with microseconds: " +
interval.milliSeconds
```

Attribute	Description
minutesOnly	Int value representing the minutes component of an interval.

Examples:

```
interval as Interval
interval = '2h20m10s'
display "minutes component of an interval: " +
interval.minutesOnly
```

Attribute	Description
minutes	Total number of minutes contained in the interval.

Examples:

```
interval as Interval
interval = '2h20m10s'
display "total minutes of an interval: "+
interval.minutes
```

Attribute	Description
monthsOnly	Int value representing the months component of an interval.

Examples:

```
interval as Interval
interval = '1Y2M3d20h'
display "months component of an interval: " +
interval.monthsOnly
```

Attribute	Description
months	Total number of months in an interval.

Examples:

```
interval as Interval
interval = '1Y2M3d20h'
display "total months of an interval: " +
interval.months
```

Attribute	Description
secondsOnly	Int value representing the seconds component of an interval.

Examples:

```
interval as Interval
interval = '1h20m35s'
display "seconds component of an interval: " +
interval.secondsOnly
```

Attribute	Description
seconds	Total number of seconds in an interval.

Examples:

```
interval as Interval
interval = '1h1m5s'
display "total seconds of an interval " +
interval.seconds
```

Attribute	Description
totalMicroSeconds	Int value representing the total number of microseconds contained in the interval.

Examples:

```
interval as Interval
interval = '20.250320s'
display "total microseconds of an interval: " +
interval.totalMicroseconds
```

Attribute	Description
yearsOnly	Int value representing the years component of an interval.

Examples:

```
interval as Interval
interval = '2Y10M15d'
display "years component of an interval: " +
interval.yearsOnly
```

Attribute	Description
years	Total number of years contained in the interval.

Examples:

```
interval as Interval
interval = '25M15d'
display "total years of an interval: " +
interval.years
```

## Interval Functions

### abs

Returns the absolute value of a given interval.

**Arguments**

- Interval - The given Interval object.

**Returns**

The absolute value of a given Interval object.

**Example**

The following example calculates an Interval by subtracting the current time from the first day of year 2000. Then, it displays the absolute value of the resulting interval:

```
year2k as Time
year2k = '2000-01-01 00:00:00'
interval = year2k - 'now'
display interval
display abs(interval)
```

**addDays**

Adds a specified number of days to an Interval object.

**Arguments**

- Interval - The interval to which days will be added.
- Int i - The number of days to be added to the Interval object.

**Returns**

An Interval object resulting from adding the specified number of days to the given Interval object.

**Example**

The following example creates an Interval variable named `holidays` and another named `newHolidays`, which is the result of adding 10 days to `holidays`. Then, it displays both the original variable and the one resulting from adding 10 days to the original variable:

```
holidays as Interval
holidays = '15d20h00m'

updatedHolidays as Interval
updatedHolidays = addDays(holidays, i : 10)

display "original holidays: " + holidays +
      "\n\n updated holidays: " +
      updatedHolidays
```

**addHours**

Adds a specified number of hours to an Interval object.

**Arguments**

- Interval - The interval to which hours will be added.
- Int i - The number of hours to be added to the Interval object.

**Returns**

An Interval object resulting from adding the specified number of hours to the given Interval object.

**Example**

The following example creates an Interval variable named `deliveryTime` and another named `newDeliveryTime`, which is the result of adding 12 hours to `deliveryTime`. Then, it displays both the original variable and the one resulting from adding 12 hours to the original variable:

```
deliveryTime as Interval
newDeliveryTime as Interval

deliveryTime = '1d00h00m'
newDeliveryTime = addHours(deliveryTime, i : 12)

display "original deliveryTime: " + deliveryTime +
        "\n\n new deliveryTime: " + newDeliveryTime
```

### **addMicroSeconds**

Adds a specified number of microseconds to an Interval object.

#### **Arguments**

##### **Interval**

The interval to which microseconds will be added.

##### **Int**

The number of microseconds to be added to the Interval object.

#### **Returns**

An Interval object resulting from adding the specified number of microseconds to the given Interval object.

#### **Example**

The following example creates an Interval variable named `retry` and another named `largerRetry`, which is the result of adding 222 microseconds to `retry`. Then, it displays both the original variable and the one resulting from adding 222 microseconds to the original variable:

```
retry as Interval
retry = '1m30.600s'

largerRetry as Interval
largerRetry = addMicroSeconds(retry, i : 222)

display "old retry: " + retry + "\n\nnew retry: " +
        largerRetry
```

### **addMinutes**

Adds a specified number of minutes to an Interval object.

#### **Arguments**

##### **Interval**

The interval to which minutes will be added.

##### **Int**

The number of minutes to be added to the Interval object.

#### **Returns**

An Interval object resulting from adding the specified number of minutes to the given Interval object.

#### **Example**

The following example creates an Interval variable named `breakTime` and another named `newBreakTime`, which is the result of adding 25 minutes to `breakTime`. Then, it displays both the original variable and the one resulting from adding 25 minutes to the original variable:

```
breakTime as Interval
breakTime = '1h20m00s'

newBreakTime as Interval
newBreakTime = addMinutes(breakTime, i : 25)

display "old break-time: " +breakTime +
        "\n\n new break-time: " + newBreakTime
```

### **addMonths**

Adds a specified number of months to an Interval object.

#### **Arguments**

- Interval - The interval to which months will be added.
- Int i - The number of months to be added to the Interval object.

#### **Returns**

An Interval object resulting from adding the specified number of months to the given Interval object.

#### **Example**

The following example defines an Interval `fishingSeason`. Then, it creates a new Interval named `newFishingSeason`, which is the result of adding a month to `fishingSeason`. Both the original interval and the result of the addition are displayed:

```
fishingSeason as Interval
fishingSeason = '1M20d'

newFishingSeason as Interval
newFishingSeason = addMonths(fishingSeason, i : 1)

display "original fishingSeason: " + fishingSeason +
        "\n\nnew fishingSeason: " + newFishingSeason
```

### **addYears**

Adds a specified number of years to an Interval object.

#### **Arguments**

- Interval - The Interval object to which years will be added.
- Int i - The number of years to be added to the Interval object.

#### **Returns**

The Interval object resulting from adding the specified number of years to the given Interval.

#### **Example**

```
licensePeriod as Interval
licensePeriod = '1Y6M'

newLicensePeriod as Interval
newLicensePeriod = addYears(licensePeriod, i : 1)
```

```
display "original license period: " + licensePeriod +
      "\n\nnew license period: " + newLicensePeriod
```

### **format**

Returns a String representation of the given interval. This String representation is based on the current locale.

#### **Arguments**

- Interval - The given Interval object.

#### **Returns**

A String representation of the given Interval based on the current locale.

#### **Example**

```
interval as Interval
interval = '2Y6M15d12h30m30s'
display format(interval)
```

### **intValue**

Returns the Int value of an Interval objects. This value represents the number of seconds in the interval.

#### **Arguments**

- Interval - The Interval object.

#### **Returns**

An Int representing the number of seconds in the given interval.

#### **Example**

The following example displays the Int representation, that is to say, the number of seconds in an interval of 1 hour, 30 minutes, and 20 seconds:

```
display intValue('1h30m20s')
```

### **max**

Returns the greater of two Interval objects.

#### **Arguments**

- Interval - The given Interval object.
- Interval b - Another Interval object.

#### **Returns**

The greater of the two Interval objects.

#### **Example**

The following example displays the greater Interval between 1 hour and 20 minutes and 1 hour and 35 minutes:

```
display max('1h20m', b : '1h35m')
```

### **min**

Returns the smaller of two Interval objects.

#### **Arguments**

- Interval - The given Interval object.
- Interval b - Another Interval object.

**Returns**

The smaller of the two Interval objects.

**Example**

The following example displays the smaller Interval between 1 hour and 20 minutes and 1 hour and 35 minutes:

```
display min('1h20m', b : '1h35m')
```

**sleep**

Causes the current BP-method to sleep for the specified number of time. Note that you cannot sleep past the current timeout. While the method is *sleeping* the timeout period is still running, therefore the sleep is applicable for the defined interval or until timeout, whatever happens first.

**Arguments**

Interval - Time to wait

**Example**

The following example pauses the execution for 5 seconds:

```
sleep('5s')
```

**Arrays****Array Overview**

An array is a collection of values of the same type. Each element of the array is identified with an *index* or a *key*. Any type that can be used to declare a variable can be used to declare an array, even another array.

**Types of arrays**

Studio supports the following types of arrays:

- [Indexed Arrays](#) on page 303
- [Associative Arrays](#) on page 304

Indexed arrays are indexed by consecutive positive integers, starting from 0 (zero).

Associative arrays may be indexed by any type, although certain types are better suited for use as indexes.

For further information about Arrays, please refer to these sections:

- [Manipulating Arrays](#) on page 305
- [Array Functions](#) on page 307
- [Array Attributes](#) on page 309
- [Array Procedures](#) on page 310
- [Mapping Array Members](#) on page 311

**Indexed Arrays**

Indexed arrays are arrays that store a set of values in a sequence of positions which are specified by an *index*, which is an integer number. PBL uses zero-based arrays, meaning that the index of the first position of the array is zero rather than one. Zero-based arrays are used in Java and Visual Basic as well.

**Declaration**

Indexed arrays are declared using square brackets:

```
ages as Int[]
```

The code above declares an indexed array named `ages`, which is of type `Int`.

### Initializing an array

You can use in-line arrays for initialization, specifying the values separated by commas:

```
ages as Int[]
ages = [23, 42, 29]
```

The code above initializes the empty array `ages`, with the integer values 23, 42, and 29.

You can add array elements at the end of the array, but must not skip any index values. That is, if the array has six elements, with indexes ranging from 0 to 5, the next assigned value must be with index 6:

```
ages as Int[]
codes = [505, 607, 404, 405, 307, 806]

codes[6] = 306
```

If you skip an index value, an *Index out of bounds* error is thrown.

### Accessing elements

The elements of an indexed array can be accessed by the index:

```
ages as Int[]
ages = [23, 42, 29]

display ages[0]
```

If you pass an index which is higher than the last available index (in the example, the last index is 2), an *Array index out of bounds* error results.

Expressions are allowed in the index, so long as they result in an integer within the array bounds:

```
codes as Int[]
codes = [505, 607, 404, 405, 307, 806]
i = 2

display codes[1 + 2]
display codes[i]
display codes[i * 2]
```

### Changing Array Values

You can change the value of any element in the array by specifying its index. This example uses a string array:

```
Names as String[]
names = ["Bill", "Ed", "Alfred"]

names[1] = "Edward"
display names
```

### Associative Arrays

Associative arrays are arrays that map (or *associate*) a set of keys to a set of values. The data type of the keys need not be an integer, so descriptive strings, for instance, may be used. Keys must be unique, but need not be contiguous, or even ordered.

### Declaration

Associative arrays are declared almost the same way as indexed arrays, with the difference that you must specify the data type of the key. For example:

```
ages as Int[String]
```

The code above declares an associative array named `ages`, which is of type `Int` and is indexed by string keys.



## Initializing an array

You can use in-line arrays for initialization. They are similar to indexed arrays, but you must specify a key for each key-value pair, since the key is arbitrary:

```
ages as Int[String]
ages = ["John" : 23, "Peter" : 42, "Mary" : 29]
```

The code above initializes the array `ages`, associating the value 23 to the key "John", the value 42 to the key "Peter", and the value 29 to the key "Mary".

You can use keys to add elements to an existing associative array. For example:

```
ages as Int[String]
ages["John"] = 23
ages["Peter"] = 42
ages["Mary"] = 29
```

The code above also initializes the empty array `ages`, and then adds associated key-value pairs. The value 23 is assigned to the key "John", the value 42 to the key "Peter", and the value 29 to the key "Mary". If a key which already exists in the array is used again with a new value, this value *replaces* the old value for that key.

## Accessing elements

The elements of an associative array can be accessed by key. The key is specified between square brackets, as an index would be for an indexed array:

```
ages as Int[String]
ages = ["John" : 23, "Peter" : 42, "Mary" : 29]

display ages["John"]
```

If you pass a nonexistent key, a null value is returned.

## Ordered arrays

An associative array can be automatically ordered by key. To do so, the array must be declared using the `ordered` option before the index declaration, as follows:

```
ages as Int[ordered String]
```

The following example will result in an ordered array, even though the array keys are initialized out of order:

```
ages as Int[ordered String]
ages = ["John" : 23, "Peter" : 42, "Mary" : 29]
for key in ages do
  display key
end
```

The keys will be displayed in ascending order, using the key data type sort order, which in this case is alphabetical.

## Manipulating Arrays

### Change Elements

To change an array element, you can just access it with its list number and assign it a new value:

```
products = ["A", "B", "C"]
products[2] = "D"
```

This changes the value of the element with the list number two (remember, arrays start counting at zero, so the element with the list number two is actually the third element.) As we changed "C" to "D", the array now contains "A", "B" and "D".

### Add Elements

Now, suppose that we want to add a new element to the array. We can add a new element in the last position by just assigning a value to the next position. If it does not exist, it is added at the end:

```
products = [ "A", "B", "C" ]
products[3] = "D"
```

Now, the array has four elements: "A", "B", "C" and "D".

It is not necessary to know the array length to add an element at the end. In order to do it, the add ([]) operator or the extend method can be used.

Example, the following method is equivalent to the previous one:

```
products = [ "A", "B", "C" ]
extend products      // add at the end
    using "D"
```

### Delete Elements

Elements can be deleted from an array by using the delete operator:

```
products = [ "A", "B", "C" ]
delete products[0]    // delete first element
```

Now, the array has two elements "B", "C".

### Find Elements

The 'in' operator can be used to check if an element is contained in an array. The following code checks whether "A" is contained in the array products:

```
products = [ "A", "B", "C" ]
if "A" in products then
    display "'products' contains the element 'A'"
end
```

Now, if you want to get the index of the first occurrence of an element:

```
products = [ "A", "B", "A", "C" ]
index = indexOf(products, "A")

if index != -1 then
    display "'A' is located at position : " + index
end
```

Last examples will show the index 0. Instead, if you want to find the last occurrence, 'lastIndexOf' can be used:

```
products = [ "A", "B", "A", "C" ]
index = lastIndexOf(products, "A")

if index != -1 then
```

```

    display "'A' is located at position : " + index
end

```

## Array Functions

### avg

Calculates the average value of the data contained in an array. Its behavior is defined for numeric element types only. The return type will be the same as the type of the array.

Arguments	<b>Array</b> array to be averaged
Returns	The average value of all numeric elements of the array. If the array is empty, returns 0 (zero). If the array contains null elements, they are ignored in the calculation. With an array containing only null elements, the function will also return 0 (zero).

The following will display 19.42:

```

array as Decimal [] = [10.49, 13.78, null, 33.99]
display avg(array)

```

### count

Counts the number of non-null elements contained in an array.

Arguments	<b>Array</b> array to be counted for non-null elements
Returns	An Integer value with the index. If the element is not found, returns -1. If the array is empty, returns -1.

The following will display 3:

```

array as Int[]=[22,33,null,55]
display count(array)

```

### indexOf

This function returns the index of the first occurrence of an element in the array. The array index starts at zero for the first element.

Arguments	<b>Array</b> array to be searched for matching elements  <b>Any</b> the element to be searched
Returns	An Integer value with the index. If the element is not found, returns -1. If the array is empty, returns -1.

The following will display 1:

```
array as String[] = ["Hello","!!!","world","!!!"]
display indexOf(array, "!!!")
```

**lastIndexOf**

This function searches the array for matching elements, and returns the index of the element's last occurrence. The array index starts at zero for the first element.

Arguments	<b>Array</b>  <b>Any</b>	array to be searched for matching elements  the element to be searched
Returns	An Integer value with the index. If the element is not found, returns -1. If the array is empty, returns -1.	

The following will display -1:

```
array as String[] = ["Hello","world","!!!"]
display indexOf(array, "happy")
```

The following will return 2:

```
array as String[] = ["Hello","world","!!!"]
display indexOf(array, "!!!")
```

The following will display 3:

```
array as String[] = ["Hello","!!!","world","!!!"]
display lastIndexOf(array, "!!!")
```

**length**

This function returns the length of the array.

Arguments	<b>Array</b>	array from which the length will be obtained
Returns	An Integer value with the number of elements in the array. If the array is empty, returns 0 (zero).	

The following will return display 4:

```
array as Int[] = [7,8,9,10]
display length(array)
```

**max**

This function returns the maximum element of the array. The element type must have a defined sorting order. The return type will be the same as the type of the array.

Arguments	<b>Array</b> array from which the maximum value will be obtained
Returns	The maximum element of the array. If the array is empty, returns a null value.

For example, the following will display "D":

```
array as String[]=["A","B","C","D"]
display max(array)
```

### min

This function returns the minimum element of the array. The element type must have a default ordering defined. The return type will be the same as the type of the array.

Arguments	<b>Array</b> array from which the minimum value will be obtained
Returns	The minimum element of the array. If the array is empty, returns a null value.

For example, the following will display 22:

```
array as Int[]=[22,33,44,55]
display min(array)
```

### sum

Calculates the sum of all the elements of the array. This function is defined for numeric element types only.

Arguments	<b>Array</b> array to be summed
Returns	The sum of all the elements of the array. If the array has no elements, returns 0 (zero).

For example, the following should display 4394:

```
array as Int[]=[112,3233,454,595]
display sum(array)
```

## Array Attributes

### first

This attribute returns the first element of an array. The datatype of the attribute depends on the data type of the array on which it is applied. For example, the first element of a String array will always be a String.

#### Returns

- the first element of the array, or null if the array is empty.

**Example**

```
array as String[]= ["Hello","World","!!!"]
display array.first
```

The expected result is "Hello".

**last**

This attribute returns the last element of an array. As with first, the datatype of the attribute depends on the data type of the array on which it is applied.

**Returns**

- the last element of the array, or null if the array is empty.

**Example**

```
array as String[]= ["Hello","World","!!!"]
display array.last
```

The expected result is "!!!".

**Array Procedures**

Array procedures operate on a given array. Unlike functions, they do not return a value. Rather, they modify the actual array which is supplied to the procedure.

**sort**

Sorts the elements of the array. The array's element type must have a defined sort order.

Arguments	<b>Array</b> array to be sorted
Result	The array with the elements sorted in ascending order. If the array contains null elements, an error will be thrown.

The following will display [2, 3, 5, 7]:

```
myArray as Int [] = [7, 3, 5, 2]
sort myArray
display myArray
```

**clear**

Clears all the elements of the array.

Arguments	<b>Array</b> array to be cleared
Result	The array with no elements.

The following will result in an empty array:

```
myArray as Int [] = [7, 3, 5, 2]
clear myArray
```

### insert

Inserts an element in the specified position.

Arguments	<p><b>Array</b></p> <p>array in which an element will be inserted</p> <p><b>Integer</b></p> <p>index after which the element is to be inserted</p> <p><b>Any</b></p> <p>a value of the same type as the array</p>
Result	The array with no elements.

The following will result in myArray being changed to [2, 9, 7, 3, 4]:

```
myArray as Int[] = [2, 7, 3, 4]
insert myArray
    using int = 1,
    value = 9
display myArray
```

### Mapping Array Members

All of the attributes of an array's element type are mapped to the array. This means that if you have an array of **MyComponent** and **MyComponent** has an attribute called **name** of String type, you can do the following:

```
array as MyComponent[]
names as String[]
// ... initialize array ...
names = array[].name
```

This has the effect of having an attribute called **name** on the array of String type.

The equivalent code without using member mapping is as follows:

```
array as MyComponent[]
names as String[]
// ... initialize array ...
for each component in array do
    names[] = component.name
end
```

The following is an example using Interval:

```
intervals as Interval[]
intervals = ['1d2h', '2d3h', '5d6h']
display intervals[].hours
```

This displays an Int array containing [2, 3, 6].

## Variables

### Variables

Provides a description of the types of variables that can be used in a PBL program.

Variables are locations in memory (and sometimes in a database) in which data can be stored. Each variable has a name, description, type, and value. Most variables only store data of one particular type. For example, if you define a variable to store a number, it would not normally be used to store a text string.

Because data types are commonly fixed, the compiler will check to ensure that you are using correct type for the values your code assigns to them. If a statement is meant to process integers, the compiler will detect when you inadvertently try to use it to process another datatype, such as a string.

### Variable Naming

Valid PBL variable names must start with an alphabetic character or an underscore character followed by zero or more alphanumeric characters. The following are valid variable names:

```
participantName = "John Doe"
iso9001 = "... "
_ugly_var_name_ = 1
```

The following PBL variable names are not valid:

```
//Variables cannot begin with numbers
4ever = true

//display is a reserved word
display = true
```

### Reserved Words

Certain names cannot be used as variable names in a PBL program. These are called reserved words. Reserved words are dependant on the current language skin. To use a reserved word as a variable name, you must escape it with the '@' character.

```
// This is legal because the reserved word
// has been escaped
@display = true

// References to it must also include
// the escape character
display = @display
```

In this example, the '@' sign is not part of the variable name. It is just a way to tell the compiler that you are not referring to the reserved word but that you want to refer (in this case) to the variable.

### Variable Declaration

The exact variable declaration syntax within a PBL program is dependant on the skin you are using. The following examples demonstrate how to declare variables using different programming styles:

Using the PBL Style:

```
name as String
temp as Any
```

Using the Visual Basic style:

```
Dim name as String
Dim temp as Any
```



Using the Java style:

```
String name;
Any temp;
```

## Variable Scope

The term *scope* is used to denote the applicability or availability of a variable within your project. For example, variables defined within the code of a method have a local scope and are therefore called *local variables*. This means that they can only be seen from within the method where they are declared.

Oracle BPM processes define special variables with special scopes, including process instance variables, project variables and predefined variables. See [Variables](#) on page 120 for details.

## Initializing Variables

### Default Values

In Studio, all variables are automatically initialized when first used if there is a suitable default value for the variable type. The following table summarizes the default values that are used for each type:

Variable Type	Default Value
Numeric	0
String	""
Any	null
Time	'now'
Interval	'0s'

### Initialization

Variables can be initialized during declaration as in the following example:

```
name as String = "Hello"
```

Variables can also be initialized after declaration by using an Assignment Statement as in the following example:

```
name as String
name = "Hello"
```

## Operators

### Operator Types Overview

An operator is a symbol that performs a function on one, two or three operands. An operator that requires one operand is called a *unary* operator. If they require two or three operands, they are called *binary* or *ternary* operators.

The following types of operators are provided in the Process Business Language:

- Arithmetic: Performs mathematical operations.
- Relational: Compares two or more values.
- Logical: Performs logical operations.

### Arithmetic Operators

The following table lists the operators that can be used in a method to perform mathematical calculations:

PBL Style	Java Style	Visual Basic Style	Description
+	+	+	Addition
-	-	-	Subtraction
*	*	*	Multiplication
/	/	/	Division
rem	%	Mod	Modulo (Remainder)

**Order of Precedence**

When multiple operators are used within an expression, the following order of precedence is followed:

- operators within parentheses are evaluated first
- multiplication, division, and modulo are evaluated from left to right
- addition and subtraction are evaluated from left to right

**Using Variables with Arithmetic Operators**

Variables of Int, Decimal, Real, Time, and Interval types may be used as operands of arithmetic operators. The result of a given expression is based on the following rules:

- If any of the operands is Real, the result is Real
- If any of the operands is Decimal and none are Real, the result is Decimal with sufficient places to hold the operation's sum, difference, multiplication, or remainder
- If both operands are Integer, the result is Integer
- Monadic '-' (change of sign) and '+' identity are supported
- The result always has at least the precision of the operand with the highest precision in the expression

**Relational Operators**

Relational operations yield a Boolean result. Relational operators must be applied to homogenous variable types. For example, comparing an integer to another integer passes the check. However, comparing an integer to a Boolean results in an error. Any variable of a numeric type is considered homogenous.

**Logical Operators**

Logical operators can be used on or between Boolean variable types to obtain a boolean result.

The following table lists the operators that are available in BPL:

Operator	Meaning	Order of Operation	Evaluation Criteria
not	Logical not	5	Negates the operand value so if the value is true, it becomes false and vice versa.
and	Logical and	6	Yields true if and only if both operands are true. Yields false if any of the operands are false. Uses evaluation by need, so if the first operand is false, the evaluation process is stopped and the result is false.

Operator	Meaning	Order of Operation	Evaluation Criteria
or	Logical or	7	Yields true if either operand is true. Yields false if and only if both operands are false. Uses evaluation by need, so if the first operand is true, the evaluation process is stopped and the result is true.

NOT Example:

```
if not found then
    // do something
end
```

AND Example:

```
if orderAmt < 200 and paymentType = "Credit" then
    // do something
end
```

OR Example:

```
if shipStatus = "Received" or shipStatus = "Pending" then
    // do something
end
```

## Statements

### Statements Overview

There are other statements, such as assignment statements, which allow you to set values to variables, and interactive statements, which allow you to interact with the current participant.

### Control Statements

Control statements are used to control the sequence of statement executions.

Top-down, sequential execution is the simplest sequence of method execution. The method starts to execute on the first line of the code, goes to the next and continues until the final statement in the code has been executed. This works fine for very simple tasks but tends to lack real-world usefulness since such a method can only handle one situation. Most programs need to be able to decide what to do in response to changing circumstances. By controlling the execution sequence according to different situations, a specific piece of code can then be used to handle more than one situation.

### Statement Timeout

To protect the Process Execution Engine against Method tasks that are not behaving as expected (such as Method with an infinite loop) and remote components, every Method has a timeout property that controls the maximum duration of Method execution. By default, this property is set to a five-minute ('5m') interval, which starts counting from the moment the Method begins to execute.

```
i = 1
while i > 0
    // ... code here
    i = i + 1
end
```

If you run the Method as displayed above without some protection mechanism, the Method would run forever (or until the engine is shut down). Besides locking the engine, infinite Method tasks also lock the instance so that no other user is able to process it. To keep locking from occurring, the timeout property invokes and ends the Method in five minutes.

The timeout property is checked when the following conditions occur:

- Method enters a loop or iteration.
- A remote component is invoked.

## Changing Method Timeouts

To change a Method timeout:

```
// Set the maximum time the Method can run

timeout = '10m'

// Execute a loop or iteration for 10 minutes at most
for each i in myArray do
    // statements
end
```

If the Method task is in an Interactive activity, consider using relay to. The relay to statement automatically ends Method execution for the activity when you expect that a user will take a certain amount of time to finish the execution. When the user finally responds, the response is routed to an alternate method designated in the relay to statement.

### Relay to statements



**Note:** From this version onwards, consider using Screenflows instead of "relay to" invocations since Screenflows have all the benefits of relay to invocations with the simplicity of process-like design.

### Controlling long running statements with relay to

When developing a Method, consider the impact the code will have on your process. For example, code in interactive activities often requires some kind of response from a human end user.

By nature, humans are unpredictable. A human end user may start his or her task in Workspace but then something may cause him or her to forget to complete the task until a later time.

Uncompleted tasks lock resources in the BPM Engine. Locked resources not only lock the method, but also decrease the scalability of the engine, which means that the engine cannot accommodate as many users as it can do under normal circumstances. To ensure that resources do not lock while waiting for end user input, you can use the relay to in your code.

When you use relay to in an interactive method, the engine immediately ends the execution of the method and does not wait for end user input. This frees the resources to handle new instances. When the end user finally enters his or her input, the engine routes the instance with the end user input to the method designated by relay to.

### Relay to example

In the following example, a simple input is requested. The relay to statement is then invoked.

```
input "Enter something in the box here: " name
    using title = "Relay To Example"
    relay to CilReachedFromRelayTo
        using relayToName = name
```

The method in the `CilReachedFromRelayTo` task is as follows:

```
// This is the standalone Method reached by an input
// statement in some Method in the process.
// Usually, the only Method you see in it is setting
// instance variable(s) from the Method's incoming
// argument variable(s).

name = arg.relayToName

display "This is the standalone Method reached from another
        Method's input statement with a \"relay to\".
        You entered: " + name
```

For further information, refer to [Using Relay To](#).

## Input Statement

Input statements allow you to invite the current participant to enter information that is required by the method's logic.

The following types are available:

- *Basic input*, which builds a simple form for entering data.
- *Interceptor input*, which intercepts one or more web pages fetching data from the different form fields.
- *BPM Object input*, which displays a presentation of a BPM Object.

Basic input and BPM Object input can be implemented with screenflows. This is strongly recommended. See Screenflow's documentation for detailed information.



**Note:** If you use the input statement in a PBL-Method in a task, when the design is checked, a warning is thrown. It is recommended to use a screenflow or the relay to option, as the input statement is not applicable if you run the process in an EJB based Engine.

## Syntax

### Basic input

```
input "field label" basicReference
    [( {option} [= {value}] , ... )]
    [ in [{valid values}] ] [, ...]
    [using
        title = "{title}",
        buttons = [{button labels}],
        cancelButton = "{button label}"
    ]
    [returning
        {selected button} = selection
    ]
```

### Interceptor input

```
input "field label" basicReference
    [ in [{valid values}] ] [, ...]
    [using
        title = "{title}",
        buttons = [{button labels}]
        htmlForm = "{intitital URL}",
        until = "{stop condition}",
```

```

        links = "{ intercept | popup | clear }",
        userControl = { true | false },
        cookies = [ { cookies } ]
    ]
    [returning
        {selected button} = selection,
        {cookie map} = cookies
    ]
]

```

The main functionality of the Web Interceptor can be simplified in these 3 topics:

- Interact with existing Web Applications.
- Allow PBL-Method access forms in HTML, JSP, ASP.
- Support any form control as Lists, Text Areas, Radio Buttons, Fields.

### How does the Web Interceptor work?

Web Interceptor basically works over HTTP(S), FTP and FILE resources.

The referenced URL (**htmlForm**) can be:

- http://...
- https://...
- ftp://...
- file://...

### BPM Object input

```

input basicReference
    [using
        selectedPresentation = "{presentation name}"
    ]
    [returning
        {selected button} = selection
    ]
]

```

### Arguments

**Attribute:** title

**Type:** String

**Description:** The input form's title.

**Attribute:** buttons

**Type:** String[]

**Description:** An array containing the labels of the buttons you want to be displayed on the form.

**Attribute:** cancelButton

**Type:** String

**Description:** button that acts ignoring all changes and avoiding the input of required fields.

**Attribute:** htmlForm

**Type:** String

**Description:** Full or relative URL to the initial page to be intercepted.

For relative files as **.html**, **.jsp** or **.asp** to be intercepted have to be copied into the installation directory, in studio/webapps/portal/**dirname**.

The **htmlForm** attribute is composed by: **http://host:port/projectname/dirname/file.[html/jsp/asp]**, where:

**host:port** is the host and port in which you run the WorkSpace.

**projectname** is the name of your project,

**dirname** is the name of the directory you created in **studio/webapps/portal/**

For example:

**htmlForm** = "**http://localhost:9595/InterceptorCase01/TestInterceptor/classRegister.html**",

**InterceptorCase01** is the name of the project,

**TestInterceptor** is the directory in studio/webapps/portal, and

**classRegisiter.html** is the html page to be intercepted.

**Attribute:** until

**Type:** String

**Description:** Stop condition that indicates the interceptor when to stop intercepting pages. It is of the form: **pattern@location**

Where **location** is a full or partial url that marks the end of interceptions and **pattern** is a string that is sought for when **location** is reached. (pattern@URL or pattern@file name)

If not found, interception continues.

Some examples for stop conditions are:

- post.asp?id=query
- http://mysite/Poster/post.asp
- Congratulations@post.asp
- Congratulations@http://mysite/Poster/post.asp

**Attribute:** links

**Type:** String

**Description:** This parameter determines what happens when the intercepted page contains links and the user clicks on one. It takes one of the following values:

- popup: the content of the link will be displayed in a new window.
- clear: the link will be removed.
- intercept: the link will be intercepted and displayed in the same window.

It will continue to intercept until the specified criteria has been met or the user hits "Stop" in the navigational control (see: **userControl** attribute.)

**Attribute:** **userControl**

**Type:** Bool

**Description:** If true, the navigational control will be displayed in the intercepted page.

One of the purposes of using Web interceptor is to set and get information from the pages you are intercepting or navigating. This basically implies that you are able to go through a sequence of HTML, JSP or ASP pages intercepting and collecting information that will later be used in the PBL-Method logic. The navigational tool allows the end user go back and forward through the pages until the stop criteria is reached.

In order to show the navigational tool in the intercepted pages you need to set "userControl" to true. If you set true to **links**, then **userControl** takes the value true automatically and the navigational toolbar is displayed.

The buttons showed in the navigational panel are:

- Go Back : Goes to the previous intercepted page.

- Finish Interceptor : Stops the interception and returns the control to the PBL-Method.
- First page: It starts again the interception from the first page.

**Attribute:** cookies

**Type:** java.Object[java.Object]

**Description:** Associative array that contains a collection of cookies used during interception. The keys and values are of String type.

**Attribute:** selection

**Type:** String

**Description:** Returns the label of the button that caused the form to be dismissed.

**Attribute:** selectedPresentation

**Type:** String

**Description:** The name of the desired presentation for the Object, if this attribute is not specified, the default presentation will be used

**Field options**

The following table contains a list of the **options** that can be passed to an input field:

Option	Required Type	Description
date	Time	displays a Time as date-only.
time	Time	displays a Time as time-only.
readonly	Any	the field is displayed, but cannot be modified.
password	String	the field is displayed as a password field.
required	Any	The field cannot be null.
textarea	String	displays an area to enter a text.

**Remarks**

For the BPM Object input, the selectedPresentation attribute is only valid if the basicReference is a BPM Object. If it is not, a field label will be synthesized and the input will behave as a basic input.

If the selectedPresentation attribute is missing, the default presentation of the BPM Object will be displayed.

When you specify a partial URL in any of the fields that take one, the URL is relative to the portal in which the input statement is displayed.

For an Interceptor input, the field name must match the name of a field in the form being intercepted. If it does not match, the variable will be left empty.

**Input Examples**

**Basic Input**

```
creditCardNo = ""
acceptedTypes = ["visa" : "Visa", "master" : "Mastercard",
    "amex" : "American Express"]
creditCardType = "visa"
firstName = ""
lastName = ""
expiration = 'now'
```



```

comments = ""
input "First Name:" firstName (required),
      "Last Name:" lastName (required),
      "Credit card type:" creditCardType (required)
      in acceptedTypes,
      "Credit card No.:" creditCardNo,
      "Expiration Date:" expiration,
      "Additional Comments:" comments (textarea)
using
  title = "Enter payment info",
  buttons = ["Ok", "Cancel"]

```

### Interceptor Input

```

googleQuery = ""
input "q" googleQuery
  using
    htmlForm = "http://www.google.com",
    links = "clear"

```

Refer to Web Interceptor for more examples.

### BPM Object Input

```

//Order is a BPM Object with a presentation called
//'auditView'
input order
  using
    selectedPresentation="auditView"

```

### Display Statement

The display statement, as its name implies, allows you to display information to the user and to get feedback based on the choice of buttons the user selected.

The **Display** can be implemented with screenflows. This is strongly recommended. See Screenflow's documentation for detailed information.



**Note:** If you use the **display** statement in a PBL-Method in a task, when the design is checked, a warning is thrown. It is recommended to use a screenflow, as the display statement is not applicable if you run the process in an EJB based Engine.

### Syntax

#### Basic Display

```

display {object}
[ using
  [title = "{title}", ]
  [type = "{error | question | warning | info | plain}",]
  [options = {options}, ]
  [default = {default button}]
]
[ returning {selected button} = selection ]

```

### BPM Object Display

```

display {fuego object}

```

```
[ using selectedPresentation = "{presentation name" ]
```

This form of display shows a BPM Object presentation as read-only.

### Arguments

The following is the list of the arguments that can be passed to a display statement:

**Argument:** title

**Type:** String

**Description:** Title of the display window/page.

**Argument:** type

**Type:** String

**Description:** Kind of display. The icon will be chosen based on this argument. The default value is "plain".

**Argument:** options

**Type:** String[]

**Description:** Array of strings containing the labels of the buttons you want to display.

**Argument:** default

**Type:** String

**Description:** Which of the buttons is returned in case the display is canceled.

**Argument:** selectedPresentation

**Type:** String

**Description:** Name of the presentation used to display a BPM Object. If left unspecified, the default presentation will be used.

### Display Examples

#### Basic Display

```
selectedButton as String
display "Should we try again?"
    using title = "Confirm",
           type = "question",
           buttons = ["Yes", "No"],
           default = "No"
    returning selectedButton = selection

if selectedButton = "Yes" then
    //Retry
end
```

#### BPM Object Display

```
//Order is a BPM Object with a presentation
//called 'auditView'
input order
    using
        selectedPresentation="auditView"
```

## Compound Statement

A *compound statement* groups other statements in a logical unit. It defines a scope and allows you to handle exceptions or execute statements that must always be executed, regardless of the outcome of the block.

### Syntax

The most basic form of a compound statement simply encloses some statements together. This is also called a code block:

```
do
    //Your statements here
end
```

You can also handle exceptions that are thrown inside the block:

```
do
    //Your statements here
on excep as Java.Exception
    //Handle Exception here
end
```

You can add some code to be executed when your block finishes, regardless of whether it has finished by exception or not:

```
do
    //Your statements here
on exit
    //Do something that must be done always, such as
    //releasing external resources
end
```

Exception and end of block handling can be combined. For example:

```
do
    //Your statements here
on ex as Java.Exception
    //Handle Exception here
on exit
    //Do something that must always be done, such as
    //releasing external resources
end
```

## About Loops and Methods

Loops and methods define special-purpose blocks, and these can also be used to handle exceptions.

### Loops

Loops are used to execute a given code block several times, usually with one or more variables which hold different values in each iteration of the loop. These values may be updated by the code block within the loop, or as a result of the loop definition itself.

For example, the following loop will iterate three times, where the name variable will successively adopt the values of "John", "Peter", and "Mary":

```
names as String[]
names = ["John", "Peter", "Mary"]
for each name in names do
    // Do something
```

```
on ex as Java.Exception
    // Handle your exceptions
end
```

which is equivalent to:

```
names as String[]
names = ["John", "Peter", "Mary"]
do
    for each name in names do
        // Do something
    end
on ex as Java.Exception
    // Handle your exceptions
end
```

## Methods

Methods define a compound statement that contains the entire body of the method. Each method is contained by an implicit do/end block. For example, the following statement:

```
on Exception
```

is semantically equivalent to:

```
do
    on Exception
end
```

This block is labeled after the method's name, so you can add exit and exception handlers directly in the method as in the following:

```
...
...
on ex as Java.Exception
    //Handle Exception here
on exit
    //Execute required code, such as releasing external resources
...
```

The choice between using an explicit do/end block or handling exceptions directly within a method depends on the scope of the method you are writing. Also, code readability should be taken into account when choosing which style to use.

## Simple Conditional Statements (if-then-else)

Describes purpose and variants of the if-then-else statement

### If-then-else

The if-then-else statement evaluates a boolean (true/false) expression. If the expression yields true, the statement block following then is executed. Else, if the expression is false, execution goes to the else statement block, if present. Execution then continues normally after the end statement.

The syntax is as follows, with the else clause being optional:

```
if <condition> then
    <statements>
[else
    <statements>]
end
```

The following example is used with display and input statements to capture end user feedback. This particular example evaluates the variable `selected` and sets the predefined action variable to `FAIL`:

```
if selected = "Cancel" then
    action = FAIL
end
```

The following example evaluates the variable `orderTotal`. If the order is greater than \$5,000, the Boolean variable `checkCredit` is set to `true`:

```
if orderTotal > 5000 then
    checkCredit = true
end
```

We can go one step further with the previous example. We can also check whether the order is a credit order or a cash order by using the logical operator and to verify the two conditions. As before, the code checks if `orderTotal` is greater than \$5,000 and now also requires that `paymentType` be set to `"Credit"`:

```
if orderTotal > 5000 and paymentType = "Credit" then
    checkCredit = true
end
```

The final example shows the use of the `or` logical operator and the `else` clause. This example checks whether the variable `lollipop` is `"cherry"` or `"raspberry"`. If so, `eat` is set to `true`. If not, `eat` is set to `false`:

```
if lollipop = "cherry" or lollipop = "raspberry" then
    eat = true
else
    eat = false
end
```

## Elseif

Some times you will need to handle more than two alternatives. To support this situation you can use the optional `elseif` clause:

```
if <condition> then
    <statements>
[elseif <condition> then
    <statements>]
...

[elseif <condition> then
    <statements>]
[else <condition> then
    <statements>]
end
```

In effect, each `elseif` concatenates two `if-then-else` statements. An arbitrary number of `elseif` blocks can be included, as well as a single optional `else` clause at the end. The example below uses the `elseif` clause and the `else` clause. This way, you can continue adding conditions indefinitely:

```
if selected = "Cancel" then
    action = FAIL
elseif selected = "Process" then
    orderStatus = "Reviewed"
    financeStatus = "Check"
else
    orderStatus = "In Review"
end
```

For situations where program flow must follow several possible options based on only one parameter, it is better to use the [Case Statement](#) on page 325 statement.

## Case Statement

Describes the case multipath conditional statement

This statement is an alternative to the [Simple Conditional Statements \(if-then-else\)](#) on page 324 conditional statement. The case construct is more efficient and easier to read when you need to execute one block of code among many, based on the value of a single parameter.

Syntax:

```
case <expression>
when <case1> then
    <statements>
[when <cases2> then
    <statements>]
...
[when <caseN> then
    <statements>]
[else
    <statements>]
end
```

There can be zero or more when-then statement blocks, though normally there will be more than one.

A given when expression can check for more than one value, where each value is placed in a comma-delimited list:

```
case x
when 1 then
    display "x is equal to one"
when 2,3,4,5,6 then
    display "x is a value between two and six"
else
    display "x is greater than six or less than one"
end
```

The else block is optional. It can be used to implement a default action if no when conditions are met.

The following case example sets the string `shortWeekday` based on the value of `dayNumber`. There are seven weekdays, so if `dayNumber` is not 1, 2, 3, 4, 5, 6, or 7, then its value is considered to be invalid.

Example:

```
case dayNumber
when 1 then
    shortWeekday = "Sun"
when 2 then
    shortWeekday = "Mon"
when 3 then
    shortWeekday = "Tue"
when 4 then
    shortWeekday = "Wed"
when 5 then
    shortWeekday = "Thu"
when 6 then
    shortWeekday = "Fri"
when 7 then
    shortWeekday = "Sat"
else
    shortWeekday = "This is not a valid weekday!"
end
```

## Bounded Loops

Describes range and key iteration bounded loops

Bounded loops allow you to execute a set of statements a number of times which is known before entering the loop. The number of times might be determined by a range or a collection.

## Range iteration (for in)

This loop iterates over an integer range. That is, on each iteration, an integer variable increments by one:

```
for <id> in <rangeStart>..<rangeEnd> do
  <statements>
end
```

 **Note:** *rangeStart* and *rangeEnd* can be expressions. The range limits are inclusive. Modifying the variable inside the loop does not have any effect on the loop's execution.


For example, this loop displays the numbers from 1 to 3, inclusive:

```
for i in 1..3 do
  display i
end
```

## Key iteration (for in)

The `for in` loop can also iterate over the *keys* or *indexes* of an array:

```
for <id> in <expression> do
  <statements>
end
```

 **Note:** *expression* must yield an array.

This example displays all the keys in the `ages` array:


```
ages as Int[String]
ages = ["John" : 23, "Peter" : 42, "Mary" : 29]

for name in ages do
  display name
end
```

## Element iteration (for each in)

The `for each in` loop iterates over the *values* of an array. Some of them may be excluded by adding a "where" clause:

```
for each <id> in <expression> [where <condition>] do
  <statements>
end
```

 **Note:** *expression* must yield an array.

This example displays all the values in the `ages` array:

```
ages as Int[String]
ages = ["John" : 23, "Peter" : 42, "Mary" : 29]

for each age in ages do
  display age
end
```

The following example shows only the ages above 25:

```
ages as Int[String]
ages = ["John" : 23, "Peter" : 42, "Mary" : 29]

for each age in ages
  where age > 25
do
  display age
end
```

## Stopping a Loop

Loop execution can be stopped by using an Exit Statement. Execution will continue from the first statement following the end of the loop.

## Unbounded Loops

Unbounded loops are useful when the programmer does not know in advance how many times the loop will be traversed.

This loop repeats an action until an associated test condition returns false or until an exit statement is executed. The condition is evaluated before entering the loop and after each iteration.

### Syntax

```
while <condition> do
    <statements>
end
```

### Example

Continue asking until the user chooses "Ok":

```
selection as String = ""

while selection /= "Ok" do
    display "Are you sure ?"
    using buttons = ["Ok", "Cancel"]
    returning selection = selectedButton
end
```

## Exit Statement

A method or a loop in a method can be interrupted by using the exit statement:

- If the loop is labeled and the exit statement refers to that label, the exit statement exits the labeled loop.
- If the name provided is the method name, the execution of that method ends.
- If no name is provided, execution exits the innermost loop of the method, or the method itself, if it is outside a loop.

Syntax:

```
exit [<label>] [when <condition>]
```



**Note:** *label* can be the method name or a labeled loop. The *condition* can be used to avoid cluttering your code with a conditional statement.

The following example finds the first e-mail containing a specific subject then exits the loop:

```
// order is a variable of type Fuego.Net.Mail

order as Mail
url as String = ""

for each mail in MailServer(url, false).messages do
    if mail.subject = "New Order" then
        order = mail
        exit
    end
end
```



```
// if there is no order to process,
// stop method execution exit when
// order is null.
```

The second example finds a participant, assigns it to the `nextParticipant` variable, and ends the method execution. The name of the method is `findParticipant`:

```
participantName = "John"

for each p in activity.role.participants do
  if p.name = participantName then
    // participant found!
    nextParticipant = p
    exit : findParticipant
    // findParticipant is the name of the method
  end
end
```

For further information refer to topics:

### Labeled Statement

Labels provide a name to identify and reference a statement. Labels are used by the exit statements to specify the statement to which the exit applies.

Note that all methods have an enclosing label which has the same name as the method.

### Syntax

```
<label> : <statement>
```

### Example

```
// order is a variable of type Fuego.Net.Mail
order as Mail
url as String = ""

mainLoop: for each mail in
  MailServer(url, false).messages do
    if mail.subject = "New Order" then
      order = mail
      exit : mainLoop
    end
  end

// if there is no order to process, stop method execution
exit when order is null
```


### Throw Statement

The throw statement lets you raise an exception, thus breaking the execution until:

- The exception is handled.
- The method finishes and the process' exception handling procedures gain control.

### Syntax

```
throw <expression>
```

 **Note:** \* **expression** must yield a `Java.Lang.Throwable`.

### Example

```
throw Java.Exception("Something is wrong")
```

### Logging Statement

The logging statement is used to log a message in the log files maintained by the Process Execution Engine. Log messages are useful to debug the behavior of your code, especially in automatic activities.

#### Syntax

```
logMessage "message to log"
  [using
    [severity = <severity>]
    [, detail = <detail>]
  ]
```

#### Severity levels

The following are severity levels that can be used with the logging statement:

- DEBUG
- INFO
- WARNING
- SEVERE
- FATAL

You can choose the severity level you want to display in the Engine logs in the Execution Console.

### Example

```
orderName = 1
customer = 1

logMessage "Order " + orderNumber + " from customer "
           + customer + "was aborted" using severity = SEVERE
```

## Regular Expressions

### Regular Expression Overview

A *regular expression* is a pattern or template for string matching. Regular expressions are written with a very specialized, powerful syntax which can perform complex string comparisons, extract desired substrings, or perform advanced search and replace operations on strings.

PBL provides support for regular expressions using syntax compatible with Perl regular expression syntax.

## Regular Expression Syntax

Regular expressions are enclosed between forward slashes:

```
{regular expression}/
```

The simplest regular expression is a single word to search for in a string. A regular expression consisting of a word matches any string that contains that word. So, the following regular expression matches any string that contains the word "hello" anywhere in the string:

```
/hello/
```

A regular expression is a pattern which is written to match single characters or multiple characters. In the case above, each one of the characters in `/hello/` is simply matching itself. The next simplest matching character is the dot ".", which will match any single character except newline "\n". For example, the expression `/c.t/` will match "cat", "cut", or any other three-character string starting with c and ending with t, so long as the middle character is not a newline.

If, however, you specifically wish to match only "cat" and "cut", you can use a character class, which is a pattern of characters within square brackets. The following matches "cat" and "cut", but no other word:

```
/c[au]t/
```

Many other characters and expressions are possible. They are described in various topics throughout this section.

## Using Regular Expressions

In PBL, you must write regular expressions between a single quotes as follows:

```
'/{regular expression}'
```

In order to actually use a regular expression to do something in PBL, you must use a function which works with regular expressions. For instance, to find out if a particular string contains a word, you need to use the `contains` function, as follows:

```
myString.contains( '/Hello/' )
```

This line of code will search for the word "Hello" in the string that is contained in `myString`. It returns true if "Hello" is found and false if it is not.

You can also apply function calls to literal strings instead of variables. For example:

```
"Hello world!".contains( '/Hello/' )
```

returns true because "Hello world!" does contain "Hello".

## Regular Expressions in Functions

In Studio, you use regular expressions by calling functions (methods) used on string objects. The following string functions support regular expressions:

Function	Purpose	Returns
<code>contains()</code>	Matches a substring contained in the string.	Bool
<code>isMatch()</code>	Matches the string completely.	Bool
<code>indexOf()</code>	Gets the first index location where the regular expression matches the string.	Int

Function	Purpose	Returns
<code>lastIndexOf()</code>	Gets the last index where the regular expression matches the string.	Int
<code>match()</code>	Attempts to match and return the substring(s) that matched the regular expression. This is useful for extraction and parsing.	String[] - The array of subexpressions (groupings) matched. When the <code>g</code> modifier is used, the array of matched occurrences is returned instead.
<code>split()</code>	Splits a string using the given regular expression as a separator.	String[] - The array of fields (pieces of the string that were separated by the given separator.)
<code>count()</code>	Gets the number of substrings (within the string) that the given regular expression matches.	Int
<code>replace()</code>	Replaces pieces of the string with new strings.	String - the new modified string.

### Search and Replace

Regular expressions can also be used to replace parts of a string by a different string. In Studio, you can do this using the `replace` function. The first argument to `replace` is the regular expression search and the second argument is the new string. For example:

```
myString = "We played 1 on 1"
myString.replace('/1/', "one")

display myString
```

In the second line, `1` is replaced by `one`. Notice that only the first occurrence of `1` is replaced when you try the code with the debugger. In order to replace all occurrences, you must use the `g` modifier. For example:

```
myString.replace('/1/g', "one")
```

The following code strips any zero digit on the left end of a string. For example, to change `005422` to `5422`, use:

```
myString.replace('/\b0+/g', "")
```

A powerful feature of the `replace` function is that you can use backreferencing in the replacing string. You do so by using `$x`, where `x` is the grouping number.

For example, to swap the first two words in a string:

```
myString.replace('/(w+) (\w+)/', "$2 $1")
```

To convert a `MM-DD-YYYY` date to `DD/MM/YYYY`, use:

```
myString.replace('/(\\d+)-(\\d+)-(\\d+)/', "$2/$1/$3")
```

To remove quotes surrounding any word:

```
myString.replace('/"(\w+)"/g', "$1")
```

## Modifiers

You may have already noticed that matching is case sensitive. This means that the regular expression will only match a substring if both the regular expression and the substring have the same upper/lowercase characters.

In order to make matching case insensitive, you need to use a modifier. Regular expression modifiers allow you to change the default behavior of the matching.

To add a modifier, you extend the basic syntax of the regular expression as follows:

```
'/regular expression/modifier(s)'
```

Each modifier is just a single character that is specified between the last slash and the quote. The modifier for insensitive case matching is `i`. So, the following regular expression matches Hello, hello, HeLIO, and hELLO:

```
'/hello/i'
```

Some other modifiers are listed in the following table:

Modifier	Meaning
i	Searches in a case insensitive manner.
g	Matches the regular expression as many times as possible. Matches all substrings globally.
s	Treats the string as a single line.
m	Treats the string as multiple lines.

More than one modifier can be used at the same time.

## Metacharacters and Character Sets

The power of regular expressions is based on the fact that they can contain special characters that perform special functions. These characters are not treated as regular characters and are not matched literally. They are known as *metacharacters*.

Some of the metacharacters that make pattern matching more generic are as follows:

```
[ ] . ( ) * ^ $ ? \
```

The `[` and `]` characters are used to specify a set of characters that you wish to match. Characters can be listed individually or a range of characters can be indicated by listing two characters separated by a dash (`-`). For example, `[aeiou]` matches any of the vowels. The set `[a-d]` is equivalent to `[abcd]`.

If you specify the `^` character right after the opening bracket, it matches the complement of the set. In other words, any character that is not in the set. For example, `[^0-9]` matches any non-digit character.

The following table lists some examples of regular expressions using sets:

Regular Expression	Strings that Match
'/[cr]at/	cat, rat
'/[0-9]/	Any digit
'/[0123456789]/	Any digit
'/[A-Z]/	Any uppercase letter
'/[0-9][0-9]/	Any two digits together (like 01, 42, 27...)

Regular Expression	Strings that Match
/[aeiou]/	Any of the vowels

Fortunately, there are some shortcuts for some of the common character sets. For example, `\d` denotes "any digit" the same way that `[0-9]` does. `\D` means "any non-digit" like `[^0-9]`. The following table lists some other common shortcuts.

Shortcut Sequence	Equivalent to
<code>\d</code>	<code>[0-9]</code>
<code>\D</code>	<code>[^0-9]</code>
<code>\w</code>	<code>[a-zA-Z0-9_]</code> Any alphanumeric character including the underscore.
<code>\W</code>	<code>[^a-zA-Z0-9]</code> Any non-alphanumeric character.
<code>\s</code>	Any whitespace character.
<code>\S</code>	Any non-whitespace character.

These shortcuts can be included inside a character class (set.) For example, `[\da-fA-F]` is a character class that will match one hexadecimal digit. The tab, new line, and return characters are specified with `\t`, `\n` and `\r`, respectively.

Another special metacharacter is the dot (`.`). A dot within a regular expression matches any character (except the `\n` character, unless the `s` modifier is used).

Special Cases

What happens if you want to search for some of the metacharacters, like `[` or `?`? You can escape these characters with a backslash (`\`) just before the character. For example, `\.`, `\`, and `\[` match a literal dot, backslash, and opening bracket, respectively.

Go back to the Method example and try these metacharacters with different regular expressions. Experiment with character classes and the shortcuts available. The only way to learn to use regular expressions is to build some.


Matching Repetitions

In the previous examples, you were only matching expressions consisting of a few generic characters and literal words. To help write more expressive patterns, you use a quantifier metacharacter. The quantifiers are as follows:

`? * + { }`

The quantifiers allow you to determine the number of repeats of a portion of a regular expression you consider a match. Quantifiers are located immediately after the character, character class or grouping that you want to match. The following table defines each quantifier and its meaning.

Quantifier	Definition
<code>a?</code>	Match 'a' one or zero times.
<code>a*</code>	Match 'a' zero or more times (any number of times.)
<code>a+</code>	Match 'a' one or more times (at least one time.)
<code>a{n,m}</code>	Match 'a' at least n times, but not more than m times
<code>a{n, }</code>	Match 'a' at least n or more times.
<code>a{n}</code>	Match 'a' exactly n times.

 **Note:** 'a' in the previous table can be any character, character class or grouping. You will learn more about groupings later but basically, you can group a part of a regular expression using parenthesis.

The following are some examples using matching repetitions:

Matching repetition	Description
<code>'/\w+/'</code>	Any alphanumeric word (one or more alphanumeric characters together.)
<code>'/-?\d+/'</code>	A number (one or more digits) optionally prefixed by a hyphen.
<code>'/[a-z]+\t\d{1,5}/'</code>	Any lowercase word, followed by a tab, followed by 1 to 5 digits
<code>'/\w+/'</code>	Any alphanumeric word (one or more alphanumeric characters together.)
<code>'/The.*dog/'</code>	Any line that is followed by anything and then dog. Examples: The nice dog The quick brown fox jumped over the lazy dog The WhateverHeredog Thedog

Now, you have enough tools to create some useful regular expressions. For example, let's build a simple regular expression to match 10 digit telephone numbers. You may start with:

```
'/d{10}/' // 10 digits (no more, no less)
```

This regular expression matches any 10 digit number but it has some weaknesses. For instance, what happens if you want to accept numbers that contain dashes (-) in between, such as 321-123-1234? In that case, you can do the following:

```
'/\d{3}-\d{3}-\d{4}/'
```

This is fine, but what if you want the dashes to be optional? Try this:

```
'/\d{3}-?\d{3}-?\d{4}/'
```

This is better, but still there are some improvements to be made. You might not want to allow a zero (0) as the first digit of the number. Thus, the first digit must be in the class [1-9] instead of `\d` as follows:

```
'/[1-9]\d{2}-?\d{3}-?\d{4}/'
```

Did you understand it? Let's study it in parts:

1. First, a digit between 1 and 9: `[1-9]`
2. Next, two digits (from 0 to 9): `\d{2}`
3. Then, an optional dash: `-?`
4. Three digits: `\d{3}`
5. Another optional dash: `-?`
6. Finally, four digits: `\d{4}`

Try it in the debugger:

```
phone as String
input "Enter your phone number:" phone
```

```

if phone.isMatch('/[1-9]\d{2}-?\d{3}-?\d{4}/') then
    display "OK, a valid phone number"
else
    display "ERROR, invalid phone number"
end

```

## Anchors

Describes purpose and use of anchor metacharacters.

When you use the contains function, it returns true if the regular expression matched anywhere in the string. However, sometimes you would like to specify where in the string the regular expression should try to match. To do this, you use *anchor* metacharacters.

## Common Anchor Metacharacters

The two most common anchor metacharacters are '^' and '\$'. The '^' anchor means match at the beginning of the line and the '\$' anchor means match at the end of the line. The following examples show how they are used:

```

display "rock and roll".contains('/and/')
    // displays true
display "rock and roll".contains('/~np~^~/np~and/')
    // displays false
display "rock and roll".contains('/~np~^~/np~rock/')
    // true
display "rock and roll".contains('/rock$/')
    // false
display "rock and roll".contains('/roll$/')
    // true
display "rock and roll".contains('/nd roll$/')
    // true
display "rock and roll".contains('/~np~^~/np~rock$/')
    // false
display "rock and roll".contains('/~np~^~/np~rock and roll$/')
    // true
display "rock and roll".isMatch('/rock and roll/')
    // true

```

The second regular expression does not match because '^' constrains and to match only if it is at the beginning of the string. The fifth regular expression does match, since the '\$' constrains roll to match only at the end of the string.

Look at the last two examples. If you use both the '^' and '\$', you mean that the regular expression must match both the beginning and the end of the string. In other words, the regular expression matches the whole string. Note that both examples are equivalent since the isMatch will always look for a complete match. The '^' and '\$' anchors are irrelevant when using isMatch.

## Difference between '^' and '\A' and between '\$' and '\Z'?

Usually, you will only use '^' and '\$', but when using the m modifier, there is a small difference. If the string contains newline (\n) characters, then the '^' and '\$' match, just after and just before, the new line. However, '\A' and '\Z' only match at the start and end of the whole string. So, using the m modifier and replacing the space in "and roll" with a newline you get the following:

```

display "rock and~np~\~/np~nroll".contains('/and$/m')
    // true
display "rock and~np~\~/np~nroll".contains('/and~np~\~/np~Z/m')

```



```
// false
display "rock and~np~\~/np~nroll".contains('/^roll$/')
// true
display "rock and~np~\~/np~nroll".contains('/~np~\~/np~Aroll~np~\~/np~Z/ ')
// false
display "rock and~np~\~/np~nroll".contains('/~np~\~/np~Arock/ ')
// true
```

The following table describes modifier behavior:

Modifier	Behavior
<i>none</i>	Default behavior. '.' matches any character except '\n'. '^' only matches at the beginning of the string and '\$' only matches at the end of the string.
<i>s</i>	Treat string as a single long line. '.' matches any character, even '\n'. '^' only matches at the beginning of the string and '\$' only matches at the end or before a new line at the end.
<i>m</i>	Treat string as a set of multiple lines. '.' matches any character except '\n'. '^' and '\$' are able to match at the start or end of any line within the string.
<i>m and s</i>	Treat string as a single long line but detect multiple lines. '.' matches any character, even '\n', '^', and '\$'. However, they are able to match at the start or at the end of any line within the string.

## Alternations

Sometimes, you need a regular expression to match different possible words or character strings. This is possible by using the alternation metacharacter (`|`). So, if you want to match any substring that contains the word `hi` or the word `hello`, then use the following expression:

```
'/hi|hello/'
```

Bear in mind that the expression tries the alternative choices from left to right trying to match the regular expression at the earliest possible point in the string. The following are some examples:

```
"black and white".contains('/black|gray|white/')
// matches black
"black and white".contains('/white|gray|black/')
// matches black. Even though white is the first
// alternative in the string, black matches
// earlier in the string.
"Bye!".contains('/b|by|bye|bye!/i')
// matches b
"Bye!".contains('/bye!|bye|by|b/i')
// matches bye!
```

The last example suggests that if some of the alternatives are prefixes of the others, they put the longest alternatives first. Otherwise, they will never match.

In some way, you can think of character classes as character alternations. So `'/[aeiou]/'` behaves like `'/a|e|i|o|u/'`.

## Grouping

Parts of a regular expression can be grouped so that they are treated as a single unit. Parts of a regular expression are grouped by enclosing them with parenthesis (). Grouping a subexpression has many uses such as for alternation on part of the whole regular expression, for repetitions, for text extraction and for backreferencing. (Extraction and backreferencing are discussed in later sections.)

Some grouping examples are displayed in the following table:

### Regular Expression

```
'/(straw|blue|rasp)berry/'
```

**String that Matches:** strawberry, blueberry, or raspberry

### Regular Expression

```
'/Blah( blah)*/'
```

**String that Matches:** Blah, Blah blah, Blah blah blah, Blah, blah, blah, blah,...

### Regular Expression

```
'/^ (a|b) /'
```

**String that Matches:** Matches either a or b at the beginning of the line (note that `'/^a|b/'` would match a at the beginning or any b anywhere).

### Regular Expression

```
'/y(es)?/i'
```

**String that Matches:** Y, y, or any case insensitive version of 'yes'.

## Extraction

In regular expressions, *extraction* refers to the storage of strings matched by one part of the regular expression with the purpose of using them elsewhere in the expression. This is very useful for parsing and for general text processing.

An extraction group is delimited by parenthesis. For each grouping, the part of the string that matches inside the parenthesis goes into a particular position within an array of matched groupings. In PBL, the extraction can be done with the match function, which returns the array of substrings for each grouping.

For example, suppose that you have a string with the current time, in *hh:mm:ss* format. You can build a basic regular expression for matching times in that format, such as `/\d\d:\d\d:\d\d/`. However, you want to know what the value of just one element, such as the hour, is. To obtain it, group each element with parenthesis. For example, `/(\d\d):(\d\d):(\d\d)/`. The following example shows how to display hours, minutes and seconds using the index numbers of the array:

```
time as String
matches as String[]
input "Enter a time (hh:mm:ss):" time

matches = time.match('/(\d\d):(\d\d):(\d\d)/')

if matches is not null then
    display "Hours: " + matches[1] + "\n" +
        "Minutes: " + matches[2] + "\n" +
        "Seconds: " + matches[3]
else
    display "Invalid time!"
end
```



**Note:** When a regular expression is matched against a string, the whole part of the string that matches is stored in position 0 (zero) of the array.

For the previous example, if you enter "12:40:23", the array will contain the following:

Position	Value
1	12:40:23
2	12
3	40
4	23

Positions are assigned to each group from left to right.

### Extraction Example

The following is a real world example of extraction. Suppose that you need to interpret a text file with lines with the following format:

*property = value*

The file can also have comment lines, which begin with the pound sign (#). A sample of the file follows:

```
# Configuration parameters
adminEmail=admin@yoursite.com
serverHost=server.yoursite.com
serverPort=12345

# some preferences
soundEnabled=false
fontSize=12

# colors
background = white
foreground = blue
```

It would be useful if you could create an associative array, for simple access to each property. For example, to get the value of the `serverPort` property defined in the file we would use:

```
port = properties["serverPort"]
```

First, you need to define the regular expression to interpret a valid line in the file. As mentioned before, lines can be in `property = value` format or they may start with a pound (#) sign. In the latter case, the line must be ignored.

The assignment lines can be matched with `/\w+=\w+/. This looks for a word (\w+) and equals sign (=) and another word (\w+).`

The following allows optional white space around the equals sign:

```
/\w+\s?=\s?\w+/
```

Now you need to group the left side word (before the equals) and the right side word (after the equals sign) so that you can extract the values:

```
/(\w+)\s?=\s?( \w+)/
```

One more detail is required. Let's force the regular expression to match the whole string. You achieve this by adding the `^` and `$` anchors:

```
/^(\w+)\s?=\s?( \w+)$/
```

The following code fragment tests the expression:

```
input "Enter a line:" line
m = line.match('/^(\w+)\s?=\s?( \w+)$/')
if m is not null then
    display "Property: " + m[1] + "\nValue: "
```

```

        + m[2]
else
    display "ERROR, invalid line!"
end

```

A comment is easy to match by using the following regular expression (remember comment lines begin with the pound sign # in the sample text file):

```
/^#.* /
```

The expression `/^#.* /` means a line beginning with # and followed by any number of characters. An alternation will allow comment lines to match and test the Method again:

```

input "Enter a line:" line

m = line.match( '/(^#.*$)|^(\\w+)\\s?=\\s?(\\w+)$/' )

if m is not null then
    if m[1] = "" then
        display "Property: " + m[3] + "\nValue: "
            + m[4]
    else
        display "Comment line found: " + m[0]
    end
else
    display "ERROR, invalid line!"
end

```

Now that you have tested the regular expression, you can remove the display statements and write the code that builds the associative array. Instead of reading the lines from an input, we read them from a file:

```

for each line in TextFile("/tmp/test.txt").lines
    m = line.match( '/(^#.*$)|^(\\w+)\\s?=\\s?(\\w+)$/' )
    if m is not null then
        // if m is not a comment
        if m[1] = "" then
            props[m[3]] = m[4]
        end
    else
        // erroneous line - ignore it
    end
end

display props

```

Replace `tmp/test.txt` with a valid file name and location before testing the code.



**Note:** The `TextFile` component contains a built-in function for creating an associative array from a properties file. This example just shows you how to use regular expressions in a real problem. If the file were compatible with a Java properties file, then the `Textfile.loadPropertiesFrom` component is the easiest solution.

The following examples show regular expression solutions to common problems.

### Example 1

Obtain the path from the filename of a fully-qualified UNIX path and filename such as `/usr/utilities/reader/readme.txt`. This requires two extractions, as follows:

```
/(.*)\\/(^[^\\/]*)$/
```

Position [1] will contain the path (`usr/utilities/reader`), and position [2] will contain the name of the file (`readme.txt`).

### Example 2

Obtain the user ID and the host name from an e-mail address such as support@bea.com. This requires two extractions, as follows:

```
/([w\.]*)@([w\.]*)/
```

Position [1] will contain the user ID (support), and position [2] will contain the host name (bea.com).

### Example 3

To extract the parts of a URL such as http://www.bea.com:80/index.html. We require the protocol, host name, port number, and resource:

```
/(\w+):\/\/\(([\^:\|]+)(:(\d+))?(\/.*)?/
```

The following values will be obtained:

Position	Value
1	http
2	www.bea.com
3	:80
4	80
5	/index.html

Note that to obtain the port number both with and without the colon, a nested extraction was used.

### Backreferencing

A substring matched by a grouping can also be referenced within the regular expression, which is known as backreferencing. Backreferencing allows you to make matches later in a regular expression depending on what matched earlier in the regular expression. You can reference a previous grouping with \x, where x is the grouping position.

The following table provides some backreferencing examples:

Regular Expression	Matching String
/(\w+\s)\w+\s\1/	The same word repeated twice. For example, the echo ha ha...
/(\w+\s)\w+\s\1/	Words with repeated parts. For example, mama, papa, coco...
/(\d)\d\d\d\1/	Any five-digit palindrome number. For example, 12321, 83638, 91119...

## Programming

### Objects

#### Objects Overview

Studio makes extensive use of components, and it includes a large library of built-in components for common tasks. You can write your own components inside Studio (Business Objects), and you can include different technologies as components in the component catalog.

Components define a type, which can be used to declare variables. All components can be used to declare a local variable or an argument variable, but not all components can be used as instance variables.

This happens because instance variables are usually persisted (a process instance variable) or transferred (a Presentable Business Object instance variable). And, for persistence or instance variable transference to work, the content of such variables must support serialization. Some components do not support serialization.

A component can be identified by its casing. Component names always begin with an uppercase letter. For further details, refer to the [General Naming Conventions](#) on page 350 topic. For further information on component usage in Studio, please refer to the following topics:

- Implementing Business Objects using BPM Objects
- Introducing BPM Objects into the BPM Project Catalog

## Creating an Object

As noted in the [Variables](#) on page 312, all variables have a type.

Variables of primitive types (such as String, Int, Real, and so on) always have a default value (as described in [Initializing Variables](#) on page 313). On the other hand, variables which have a non-primitive type have special initialization rules.

In this section, we will discuss how to explicitly initialize non-primitive variables.

## Constructors

For initialization, we can group components in two categories:

- Instantiable components
- Components that must be obtained from another component

The difference between the two categories is that the first has a special method called a *constructor*, which is used to create new instances of the component. The second does not.

Constructors are methods that are named after the component, and may or may not have arguments. If the constructor of a component does not have arguments, it is called the *default constructor*.

The syntax to initialize a variable is the following:

```
variable = [Module.]ComponentName([[{argument name}:{value}[, ...]])
```

Note that the names and types of arguments depend on the component and on the constructor you are calling.

Consider the following example:

```
configFile as TextFile
configFile = TextFile(name : "/home/config.props")
for each line in configFile.lines do
    //Process the lines...
end
```

In the example above, on the first line, a local variable named `configFile` of type `TextFile` is declared, and on the second line it is initialized using `TextFile`'s constructor, passing a file name as an argument.

## Duplicating an Object

Describes the clone function, used to duplicate objects.

## Clone Function

Sometimes you want to create an exact copy of an object.

Simply assigning a component to a variable *does not* create a copy of it. Rather, it creates an additional *reference* to the same object. If any property of the original object changes, the new reference will also show these changes, since it is still pointing to the same object.

In order to actually create a new duplicate of the object, you can use the clone function:

```
// Create an instance for each
// participant in the role

for each person in activity.role.participants do
    copy = clone(this)
    copy.participant.next = person
end
```

### Function Behavior

The clone function behaves differently depending on how the object to be cloned is implemented. To be able to respond to different conditions, the function follows the following steps:

1. If the object you are trying to clone has a method named clone and implements the interface Cloneable, that method is used to obtain a copy.
2. If the object implements the Serializable interface, it attempts to serialize it and deserialize it to obtain a copy of the object.
3. Otherwise, it attempts to dynamically create a copy of it.

### Current and Default Instances

Object instance behavior under PBL

#### Default Instance

Oracle BPM has the concept of a default instance, which is an instance associated to a component.

Only components that have default constructors have default instances. Such instances are accessible within the method scope. That is, a default instance that has been created while running a specific method only exists through that method's execution.

For example:

```
show Menu
    using entries = ["Apples", "Oranges", "Chocolate"],
        title = "Which do you like best?"
```

In the example above, the reference to the component Menu is using its default instance, that is, an instance is automatically created the first time that a reference to Menu appears.

#### Current Instance

Typically, when you want to refer to your current instance, you use the keyword `this` (or `Me`, in Visual Basic style):

```
update this using date = 'now'
```

Suppose that the code above belongs to a component named `MyComponent`. In this case, you could also write it as follows:

```
update MyComponent using date = 'now'
```

Here, the default instance of `MyComponent` is the same as the `this`, so the current instance is used as the default instance.

### Object Cleanup

Studio automatically releases the memory used by components when they are no longer used. It is not necessary to 'release' or 'clean' the components used in a method. However, there are certain components

that require some kind of cleaning before ending the execution. They must be cleaned by using an 'exit' block to ensure that they are always cleaned up. For example:

```
do
    // use components here
on exit
    // clean used components here
end
```

Note that the code enclosed in the **on exit** (Java style: **finally**, VB style: **Finally**) part of the block is executed even if an exception occurs. Next, after the execution, the original exception is thrown, unless it is masked by an exception thrown in the **on exit** block.

## Code Conventions

### Code Conventions Overview

Provides a general description of code conventions

In any computer language, *code conventions* are a set of rules that should be followed when writing program code. They are called conventions because they are not enforced by the compiler, as they are not a part of the language syntax itself. For example, a variable can be named `lastName` or it can be named `lstn01`. The first choice is easier to read for humans, but to the compiler either is valid. Think of code conventions as a set of best practices, which under normal circumstances should be adhered to as closely as possible.

As a general rule, the purpose of code conventions is to improve readability and prevent bugs. To the extent that everybody adopts the same conventions when programming, each individual will be able to understand the work of others, and fewer mistakes will be made. This is even the case when a single individual reads his own code many months or years after having written it.

One way to improve readability is to include comments in your code. On the other hand, for the most part code conventions guide the way the code itself is structured. There are a number of ways to make code more readable:

- Adhere to variable and object naming conventions
- Use explicit variable names
- Indent code according to depth within code blocks
- Express logical statements in the simplest way possible
- Use whitespace to separate program segments

The Studio editor can help you adhere to some of these code conventions using two commands: **Indent** and **Refactor**.

### Improving Code Readability

#### Nested conditional statements

Nested conditional statements are automatically grouped in one statement with both conditions.

#### Before

```
a as Int

if a > 2 then
    if a < 10 then
        a = 5
    end
end
```



**After**

```
a as Int
if a > 2 and a < 10 then
    a = 5
end
```

**Identifiers for Exceptions**

Identifiers for exception handlers are automatically added when they are not specified.

**Before**

```
message as String
do
    message = "Ok"
on Exception
    message = Exception.message
end
```

**After**

```
message as String
do
    message = "Ok"
on e as Exception
    message = e.message
end
```

**Bounded loop instead of unbounded loop**

Unbounded loops are converted to bounded loops when possible.

**Before**

```
i = 0
while i <= 10 do
    i = i + 1
end
```

**After**

```
for i in 0..10 do
end
```

**Conditional Exit**

Conditional statements with an exit statement are transformed to conditional exits.

**Before**

```
array as Int[] = [10, 20, 30]
for each e in array do
```

```

    if e = 20 then
        exit
    end
end

```

**After**

```

array as Int[] = [10, 20, 30]

for each e in array do
    exit when e = 20
end

```

**Redundant negation**

Redundant negations are removed.

**Before**

```

a as Int = 5

if not a != 2 then
end

```

**After**

```

a as Int = 5

if a = 2 then
end

```

**Conditional statement inside else blocks****Before**

```

a as Int = 2
if a < 2 then
    a = 2
else
    if a > 5 then
        a = 5
    end
end
end

```

**After**

```

a as Int = 2

if a < 2 then
    a = 2
elseif a > 5 then
    a = 5
end
end

```

**Check for null value**

**Before**

```
s as String
if s != null then
end
```

**After**

```
s as String
if s is not null then
end
```

**Right order for 'is not'****Before**

```
s as String
if not s is null then
end
```

**After**

```
s as String
if s is not null then
end
```

**Redundant equality****Before**

```
found as Bool
if found = false then
end
```

**After**

```
found as Bool
if not found then
end
```

**Explicit argument names****Before**

```
open TextFile using "", ""
```

**After**

```
open TextFile using name = "",
                  lineSeparator = ""
```

**Unneeded parenthesis**

This refactory is applied to 'if' and 'while' conditions in Process Business Language (PBL).

**Before**

```
a as Int = 2

if (a > 2) then
end
```

**After**

```
a as Int = 2

if a > 2 then
end
```

**Legacy multi path conditional statements****Before**

```
a as Int = 2

switch a in
  case 2:
    display "Two"
  case 4:
    display "Four"
end
```

**After**

```
a as Int = 2

case a
when 2 then
  display "Two"
when 4 then
  display "Four"
end
```

**Functions**

Functions are rewritten using functional syntax.

**Before**

```
a as Int
a = a.abs()
```

**After**

```
a as Int
a = abs(a)
```

**Wrong symbols**

In PBL, some invalid symbols (e.g: &&, ||, !=, ==) are accepted but they are automatically fixed when the code is rewritten.

**Before**

```
a as Int = 2
b as Int = 4

if ((a > 2 && a < 10) || b == 4) then
end
```

**After**

```
a as Int = 2
b as Int = 4

if (a > 2 and a < 10) or b = 4 then
end
```

**Misspelled member names****Before**

```
Open TextFile using name = "",
                    lineSeparator = ""

Mail.content_type = ""
```

**After**

```
open TextFile using name = "",
                  lineSeparator = ""

Mail.contentType = ""
```

**Methods equals() and toString()****Before**

```
if object.equals(this) then
    string = object.toString()
end
```

**After**

```
if equals(object, arg1 : this) then
    string = toString(object)
```

```
end
```

## General Naming Conventions

Names representing types or modules must be nouns and they must be written in mixed case starting with upper case:

```
Line, FilePrefix
```

Variable names must be in mixed case starting with lower case:

```
line, filePrefix
```

Makes variables easy to distinguish from types and effectively resolves potential naming collision as in the declaration `Line line`.

Names representing constants (or enum values) should be all uppercase using underscore to separate words:

```
MAX_ITERATIONS, RED, MONDAY
```

Names representing methods must be verbs and they must be written in mixed case starting with lower case:

```
find(), computeTotalWidth()
```

This is identical to variable names but, in Studio, methods are already distinguishable from variables by their specific form.

Abbreviations and acronyms should not be uppercase when used as a name:

```
exportHtmlSource();    // NOT: exportHTMLSource();
openDvdPlayer();       // NOT: openDVDPlayer();
```

Using all uppercase for the base name will trigger conflicts with the naming conventions given above. A variable of this type should be named `dVD`, `hTML`, and so on, which is obviously not very readable. Another problem is illustrated in the examples above. When the name is connected to another, the readability is seriously reduced. The word following the acronym does not stand out as it should.

Variables should not have prefixes or suffixes.

Given the fact that the PBL Editor has already colored variables in a different way depending on their scope (local, instance, and so on), it is not necessary to add prefixes:

```
length as Int
this.length = length
```

Generic variables should have the same name as their type:

```
assignTopic (topic : Topic)

NOT assignTopic (value : Topic)
NOT assignTopic (aTopic : Topic)
NOT assignTopic (x : Topic)
```

Reduces complexity by reducing the number of terms and names that are used. Also, this makes it easier to deduce the type given a variable name only.

If, for some reason this convention does not seem to fit, it is a strong indication that the type name is badly chosen.

Non-generic variables have a role. These variables can often be named by combining role and type:

```
startingPoint as Point
centerPoint as Point
loginName as Name
```

All names should be written in English.

```
fileName NOT nomArchivo
```

English is the preferred language for international development.

Variables with a large scope should have long names, and variables with a small scope can have short names.

Scratch variables used for temporary storage or indexes are best kept short. A programmer reading such variables should be able to assume that its value is not used outside a few lines of code. Common scratch variables are:

integers	i, j, k, m, n, i1, i2
booleans	b,b1,b2
reals	x,y,z,w
Strings	s,str

The name of the object is implicit and should be avoided in a method name.

```
line.length
NOT line.lineLength
```

The latter seems natural in the class declaration but it proves superfluous in use, as shown in the example.

### Specific Naming Conventions

Describes naming conventions

The term "compute" can be used in methods in which something is computed.

```
computeAverage valueSet
computeInverse matrix
```

Gives the reader the immediate clue that this is a potential time-consuming operation and, if used repeatedly, he/she might consider caching the result. Consistent use of the term enhances readability.

The term "find" can be used in methods where something is looked up.

```
findNearest Vertex
findMinElementIn matrix
NOT getMinElementIn matrix
```

Gives the reader the immediate clue that this is a simple look up method with a minimum of computations involved, *but* more expensive than a simple getter. Consistent use of the term enhances readability.

The term "initialize" can be used where an object or a concept is established.

```
initializeFontSetFor Printer
```

The American spelling of "initialize" should be used instead of the English "initialise". Abbreviation of "init" must be avoided.

"n" prefix should be used for variables representing a number of objects.

```
nPoints, nLines
```

The notation is taken from mathematics, where it is an established convention to indicate a number of objects.

If "number of" is the preferred statement, numberOf prefix can be used instead of just n. The **num** prefix must not be used.

The "No" suffix should be used for variables representing an entity number.

```
tableNo, employeeNo
```

The notation is taken from mathematics, where it is an established convention for indicating the number of an entity.

Complementary names should be used for complementary concepts or actions:

- add/remove
- create/destroy
- start/stop
- insert/delete
- increment/decrement
- old/new
- begin/end
- first/last
- up/down
- min/max
- next/previous
- old/new
- open/close
- show/hide
- get/set

Reduce complexity by symmetry, and avoid abbreviations in names wherever possible. For example, use `computeSalary`, rather than `compSal`.

Exception classes should be suffixed with `Exception`:

```
AccessException
```

Exception classes are really not part of the main design of the code. Naming them like this makes them stand out relative to the other classes.

Functions (methods returning an object) should be named after what they return, and procedures (void methods), after what they do.



Increase readability. Makes it clear what the unit should do and, especially, what it is not supposed to do. Again, this makes it easier to keep the code free from causing undesired side effects.

## Negation

Negated boolean variable names must be avoided. For example, `isError` is better than `isNotError`.

The reason is that a readability problem arises when the logical not operator is used, and double negative arises. It is not immediately clear what `not isNotError` means.

## Abbreviations

When considering the use of an abbreviation, think of which kind of word you are using. Common words listed in a language dictionary should almost never be abbreviated. Avoid writing *pt* instead of *point*, *comp* instead of *compute*, *init* instead of *initialize*, and so forth.

On the other hand, there are also domain-specific phrases that are more naturally known through their acronym or abbreviation. These phrases should be kept abbreviated. For example, don't write: *Hypertext Markup Language* instead of *HTML*, or *Central Processing Unit* instead of *CPU*.

## Creating Statements

### Variables



**Tip:** Variables should be initialized where they are declared and they should be declared in the narrowest possible scope. This ensures that variables are valid at any time.

Sometimes, it is impossible to initialize a variable to a valid value where it is declared. In these cases, it should be left uninitialized rather than initialized to some phony value.



**Tip:** Variables must never have dual meaning.

Enhance readability by ensuring that all concepts are uniquely represented. Reduce the chance of error by side effects.



**Tip:** Variables should be kept alive for as short a time as possible.

By keeping the operations on a variable within a narrow scope, it is easier to control the effects and side effects of the variable.

### Loops



**Tip:** Loop variables should be initialized immediately before the loop.


```
ready as Bool = true
while ready do
    //Do something
end
```

Not:

```
ready as Bool = true


// Other stuff....

while ready do
    //Do something
end
```

 **Tip:** The use of break and exit should be minimized.

These statements should be used only if they prove to give a higher readability than their structured counterparts.

## Conditionals

 **Tip:** Complex conditional expressions should be avoided. Introduce temporary boolean variables instead.

For example, the following code:


```
if (elementNo < 0) or (elementNo > maxElement)
    or elementNo = lastElement then
    //Do something
end
```

Should be replaced by:

```
isFinished as Bool = (elementNo < 0) or (elementNo > maxElement)
isRepeatedElement as Bool = elementNo = lastElement

if isFinished or isRepeatedElement then
    //Do something
end
```


By assigning boolean variables to expressions, the program gets automatic documentation. The construction will be easier to read and debug.

 **Tip:** The nominal case should be put in the if-part and the exception in the else-part of an if statement.

```
isError as Bool = readFile (fileName)

if not isError then
    //Do something
else
    //Handle the error
end
```

Makes sure that the exceptions do not obscure the normal path of execution. This is important for both the readability and performance.

 **Tip:** Executable statements in conditionals must be avoided.


```
file = openFile (fileName, "w")
if file /= null then
    //Do something
end
```

Not:

```
if (file = openFile (fileName, "w")) is not null then
    //Do something
end
```

Conditionals with executable statements are very difficult to read. This is especially true for new programmers.

## Miscellaneous


 **Tip:** The use of magic numbers in the code should be avoided.

If the number does not have an obvious meaning by itself, the readability is enhanced by introducing a named constant instead.

```
d = s / SECONDS_PER_DAY
timeout = DEFAULT_TIMEOUT
```

Not:

```
d = s / 86400
timeout = 1000
```

 **Tip:** Real and Decimal constants should always be written with a decimal point and at least one decimal:

```
total = 0.0
speed = 3.0

sum = (a + b) * 10.0;
```


Not:

```
total = 0;
speed = 3;

sum = (a + b) * 10;
```

This emphasizes the different nature of integer and floating point numbers, even if their values might happen to be the same in a specific case.

Moreover, as in the last example above, it emphasizes the type of the assigned variable (sum) at a point in the code where this might not be evident.

 **Tip:** Real and Decimal constants should always be written with a digit before the decimal point.

```
total as Real = 0.5f
```

Not:

```
total as Real = .5f
```

The number and expression system in Studio is borrowed from mathematics and one should adhere to mathematical conventions for syntax wherever possible. In addition, 0.5 is much more readable than .5. There is no way it can be mixed with the integer 5.

## Code Layout and Comments

### Indentation

Indentation enhances readability, particularly within loops and conditional statements. Under PBL, standard practice is to indent four spaces per level.

```
for i in 1..10 do
```

```

    a[i] = 0
end

```

To facilitate this, the Tab key inserts four spaces within the code editor.

The if-then-else class of statements should have the following form:

```

if condition then
    // statements
end

```

or:

```

if condition then
    statements
else
    statements
end

```

or:

```

if condition then
    statements
elseif condition then
    statements;
else
    statements;
end

```

Not:

```

if condition
then
    statements
end

```

And not:

```

if condition then statements end

```

The chosen approach is considered to be better since each part of the if-else statement is written on separate lines of the file. This should make it easier to manipulate the statement, for instance, when moving else clauses around.

A bounded loop should have the following form:

```

for i in set do
    //statements
end

```

An unbounded loop should have the following form:

```

while condition do
    //statements
end

```

A Multipath conditional statement should have the following form:

```
case condition
when ABC
    // statements
when DEF
    // statements
else
    // statements
end
```

A Compound statement should have the following form:

```
do
    statements;
on e as Exception exception
    statements
end
```

or:

```
do
    statements;
on e as Exception
    statements
on exit
    statements
end
```

### White Space in Expressions

- Conventional operators should be surrounded by a space character.
- Reserved words should be followed by a white space.
- Commas should be followed by a white space.
- Colons should be followed by a white space.
- Semicolons for statements should be followed by a space character.

```
a = (b + c) * d
```

These rules make the individual components of the statements stand out and enhance readability. It is difficult to give a complete list of the suggested use of white space in Studio code. However, the examples shown above should give a general idea.



**Note:** Logical units within a compound statement should be separated by one blank line. This enhances readability by introducing a white space between logical units of a block.

### Comments




**Remember:** Tricky code should not be commented on but rewritten.

In general, the use of comments should be minimized by making the code self-documenting through appropriate name choices and an explicit logical structure.



**Tip:** All comments should be written in English.

In an international environment, English is the preferred language.

 **Tip:** Minimize the use of multi-line comments.

```
// Comment spanning
// more than one line
```

Since nested multi-line comments are not supported, using single line comments ensures that it is always possible to comment out entire sections of code for debugging purposes, among others.

 **Remember:** Comments should be indented in relation to their position in the code.


```
while true do
    // Do something
    something()
end
```

Not:

```
while true do
// Do something
    something()
end
```

This is to prevent comments from breaking the logical structure of the program.

## Files

 **Tip:** File content must be kept within 100 columns.

 **Tip:** The incompleteness of split lines should be made obvious.

```
totalSum = a + b + c +
           d + e

function (param1, param2,
         param3)

passingText ("Long line split" +
            "into two parts.")
```

Split lines are required when a statement becomes too wide to read comfortably, or exceeds the column limit given above. It is difficult to provide strict rules for how lines should be split, but the examples above can serve to illustrate the guidelines shown below.

In general it is good practice to:

- Break after a comma.
- Break after an operator.
- Align the new line with the beginning of the expression on the previous line.

## Embedded SQL

### Embedded SQL Overview

Introduces embedded SQL

Studio supports embedded SQL (ANSI-92 Entry Level), written directly in the code. This section describes the supported syntax and provides some examples.



**Note:** In the syntax and examples in the topics of this section, SQL keywords appear in all-caps. This is an SQL convention and it is not required by Studio. However, it is a useful convention which helps differentiate SQL commands from regular code.

For further information about SQL Components, refer to [SQL Components](#) on page 174.

### SQL Operators

Describes common SQL operators

SQL operators allow you to control query selection criteria and the values returned. The following is a list of operators supported in PBL.

#### LIKE operator

The LIKE operator searches for strings that match a specific pattern. The percent sign "%" matches any string and the underscore "\_" matches any single character. The following example returns any row where fname starts with "J":

```
for each e in
  SELECT *
  FROM employees
  WHERE fname LIKE "J%"
do
  // do something here
end
```

#### Concatenation operator (||)

In SQL statements, the || operator is used to concatenate two values of any type. For example:

```
for each e in
  SELECT lname || ", " || fname AS fullname
  FROM employees
do
  display "full name: " + e.fullname
end
```



**Note:** In PBL and Java style, the || symbol means "or" and it is used in conditional expressions. For further information, please see [Logical Operators](#) on page 314.

#### IN operator

The IN operator matches a column value against a set of literal values:

```
column_name IN (<value1>, <value2>, ...)
```

For example:

```
for each e in
  SELECT lname, fname
  FROM employees
  WHERE salary IN (20000, 25000, 30000)
do
```

```

        display "name: " + e.lname
    end

```

This statement is equivalent to the following:

```

for each e in
    SELECT lname, fname
    FROM employees
    WHERE salary = 20000 or salary = 25000 or salary = 30000
do
    display "name: " + e.lname
end

```

### IS operator

The IS operator locates a record that does or does not have a null value for a particular column:

```
<column_name> IS [NOT] NULL
```

For example:

```

for each e in
    SELECT lname, fname
    FROM employees
    WHERE address IS NOT NULL
do
    display "name: " + e.lname + ", address: "
        + e.address
end

```

### BETWEEN Operator

The BETWEEN operator allows you to select records that are between two values:

```
<column_name> [NOT] BETWEEN <value1> AND <value2>
```

The expression `a BETWEEN b AND c` is equivalent to `a >= b AND a <= c`. For example:

```

for each e in
    SELECT lname, fname
    FROM employees
    WHERE salary BETWEEN 20000 AND 30000
do
    display "name: " + e.name + ", salary: "
        + e.salary
end

```

### SQL Keywords

Lists SQL keywords

Some words in an SQL statement have a special meaning and cannot be used as regular identifiers. These keywords include the following:

```

ALL
AND
AS
ASC
AVG
BETWEEN
BY
COUNT
DELETE
DESC
DISTINCT
FROM
GROUP

```



```
HAVING
IN
INSERT
INTO
IS
LIKE
MAX
MIN
NULL
OR
ORDER
SELECT
SET
SUM
UPDATE
VALUES
WHERE
```



**Note:** These keywords can be used as regular identifiers *outside* SQL statements. Also, in compliance with SQL standards, keyword case is ignored within SQL statements, so `SELECT`, `Select`, and `select` are all accepted.

### INSERT Statement

Describes the SQL INSERT statement and common options

The `INSERT` statement is used to add one or more rows to a table. Columns may be specified by position or by name. If specified by position, the order of the values must match the position of the corresponding columns in the table. If columns are specified by name, they may be listed in any order.

In general, it is recommended practice to specify columns by name, since the table may be modified and column positions may change, breaking your code. Name references are position independent, and a `SELECT` statement with column names is more explicit and easier to read.

Columns not specified in the column list are set to their default values or to null. If the column value cannot be set to null (if it is defined as `NOT NULL` in the database) and it has no default value, an error will result and the `INSERT` will fail.



**Note:** In the syntax and examples below, SQL keywords appear in all caps. This is an SQL convention and it is not necessary in Studio. However, it is a useful convention to differentiate SQL statements from regular code.

There are two alternative ways to supply data for the `INSERT` operation. One is to specify a list of values directly:

```
INSERT INTO <table_name> [(<col_name1>, <col_name2>, ...)]
VALUES (<value1>, <value2>, ...)
```

The other alternative is to specify a `SELECT` query. In this case, the rows obtained from this query will be inserted into the table, so long as the column values from the query match the column values required by the `INSERT`:

```
INSERT INTO <table_name> [(<col_name1>, <col_name2>, ...)]
<select statement>
```

The following example specifies a set of values and will insert one row:

```
INSERT INTO employees(fname, lname, salary)
VALUES ('John', 'Smith', 25000)
```

The value set can also include expressions:

```
firstname = 'John'
salary = 20000
```

```
INSERT INTO employees(fname, lname, salary)
VALUES(firstname, "Smith", salary + 5000)
```

The following example uses INSERT with a SELECT statement:

```
INSERT INTO students
SELECT *
FROM employees
WHERE salary > 30000
```

## UPDATE Statement

Describes purpose and syntax of the UPDATE statement

The UPDATE statement modifies a set of field values in each row which satisfies a given search condition. If no row matches the condition, the UPDATE will have no effect.



**Note:** In the syntax and example sections below, SQL keywords appear in all caps. This is an SQL convention and it is not required by Studio. However, it is useful to help differentiate SQL commands from regular code.

Syntax:

```
UPDATE <table_name>
SET <column_name1> = <value-expression1>,
    <column_name2> = <value-expression2>,
    ...
[WHERE <condition>]
```

For example, the following UPDATE increases the salary by 10% for all employees who earn less than \$25,000:

```
UPDATE employee
SET salary = salary * 1.1
WHERE salary < 25000
```



**Note:** If the WHERE condition is not specified, *all* the rows will be updated.

## DELETE Statement

Describes purpose and syntax of the SQL DELETE statement

DELETE removes all rows that satisfy a given condition from a table:

```
DELETE FROM <table_name>
[WHERE <condition>]
```

The following example deletes all employees whose first name is "John" and last name is "Smith":

```
DELETE FROM employees
WHERE fname = "John" and lname = "Smith"
```

## SELECT Statement

Describes the SELECT statement and basic query options

The SELECT statement finds and retrieves rows, columns, and derived values from one or more tables of a database. The SELECT statement is flexible, with many options, and accepts column specifications, search conditions, ordering instructions, and other parameters. The powerful and complex SELECT statement is a core feature of SQL and a thorough description of it well exceeds the scope of this document. This section outlines basic SELECT syntax and clauses which suffice for most simple queries.

## Syntax

A SELECT operation may retrieve one or many records, so in PBL it is commonly placed within a for each loop, as follows:

```
for each <variable> in
  SELECT [DISTINCT | ALL] <column1>, <column2>, ...
  FROM <table1>, <table2>, ...
  [WHERE <condition>]
  [GROUP BY <grouping-column1>, <grouping-column2>, ...]
  [HAVING <group-selection-condition1>]
  [ORDER BY <ordering-col1> [ASC | DESC],
            <ordering-col2> [ASC | DESC], ...]
do
  // ...
end
```

The columns to be retrieved are delimited by commas or, alternatively, an asterisk (\*) may be used to retrieve all columns from every table queried. It is recommended that you specify each column you need rather than retrieving all of them, as this will improve performance, substantially if the table is large and contains many columns you do not need.

You can request that a column be returned under an alias by using the AS clause:

```
SELECT clientId, fn AS firstName FROM clients
```

This selects `clientId` and `fn` from the database, but it will return the columns as `clientId` and `firstName`. In this way, you will be able to access the column using `firstName` in your code rather than `fn`, so it will be easier to read.

A column can also be an expression or an aggregate function. Aggregate functions combine values from every row into a single value, such as a sum or an average, and are discussed below.

You may use the ORDER BY clause to sort the results of the query according to a given value. Ordering may be ascending (ASC), or descending (DESC). Ascending order is the default and need not be specified. You can order by one or more columns, delimited by commas. Sorting is first done on the first ORDER BY column, and subsequent ORDER BY columns are used when the previous column contains equal values.

The following example displays the name of every employee with a salary higher than 25,000:

```
for each e in
  SELECT *
  FROM employees
  WHERE salary > 25000
  ORDER BY lname
do
  display "employee name: " + e.lname + ", " + e.fname
end
```

## Using Aggregate Functions

The following example selects the maximum salary in the employee table using the MAX function. The `row.1` term is used to specify the first column. The variable `salary` is used to store the result of the maximum salary in the table:

```
for each row in
  SELECT MAX(salary)
  FROM employees
do
  salary = row.1
end
```

The following example returns the average salary of employees grouped by depnumber, but does not return employees where depnumber is equal to 3 or 4, or cases where 5 or fewer employees have the same depnumber value:

```
for each e in
  SELECT depnumber, COUNT(*), AVG(salary)
  FROM employees
  WHERE depnumber != 3 and depnumber !=4
  GROUP BY depnumber
  HAVING COUNT(*) > 5
  ORDER BY depnumber
do
  // ...
end
```

### Stored Procedures

Describes support for stored procedures

Any procedure that you have developed in your database system (such as Oracle or Microsoft SQL Server) is added to the catalog during introspection. The stored procedure can be treated as a method and used in your code. Any procedure that uses vendor specific features, such as rowtype in Oracle, is not supported. Only standard SQL procedures are added to the catalog.