OpenEdge Business Process Server: Application Developer's Guide
# Table of Contents

## Preface
- About this documentation ................................................................. 11
- User types .......................................................................................... 11
- Information on documentation ............................................................. 12
- Conventions used in this manual .......................................................... 13
- Product support contact information .................................................... 14

## Chapter 1: OpenEdge Business Process Server overview
- Business Process Server components .................................................... 19
- How Business Process Server works ..................................................... 21
- Business Process Server user types ...................................................... 22

## Chapter 2: Introducing Business Process Server application development
- Outlining Business Process Server application requirements .................... 25

## Chapter 3: Developing an application
- Application development stages ......................................................... 28
- Constraints when naming applications ............................................... 28
- Using a sample application to demonstrate application development .......... 30
- Defining the process flow of an application .............................................. 30
- Using dataslots to define the information flow ........................................ 31
- Using performers in activity worksteps ............................................... 37
- Selecting workstep interfaces ............................................................. 38
- Using the BPM Workflow Web interface generator .................................. 38
- JSPs generated for a default presentation ................................................. 39
- Using BPM Workflow JSP tags ............................................................. 39
- Using the getLabel tag ....................................................................... 43
- Determining duration, due date, and overdue periods ............................. 43
- Setting overdue actions in a workstep ................................................... 44
- Using JavaScript in a workstep ........................................................... 47
- Integrating JavaScript .......................................................................... 48
- Debugging the JavaScript code using writeLog ....................................... 55
- Publishing the application .................................................................. 55
- Installing the application ..................................................................... 56
- Running the application ...................................................................... 56
- Removing and reinstalling an application .............................................. 56
Chapter 11: Understanding the Business Process Server repository...151
Understanding the BPM Process Store database...152
Business Process Server event management...152
  Event and CommonEvent class...153
  Schema for event tables...155
  Event database keys...157
Schema for process tables...164
  Process template tables...164
  Process instance tables...165
  Meta data information tables...166
  Workstep tables...168
  Work item tables...170
Dashboard database tables...170
  STAT_APP_AVG_DURATION table...171
  STAT_APP_PRIORITY table...171
  STAT_APP_DUE_INFO table...172
  STAT_APP_STATUS_HISTORY table...172
  STAT_WI_DUE_INFO table...173

Chapter 12: Developing management applications to control infopads.175
Creating a custom HTML application...176
Creating a Java applet...178
  Using the setInfopadRequest service...179
  Using the requestInfopadValues service...180
  Using the updateInfopad service...180
Chapter 13: Developing Balanced Scorecard applications..............183

Understanding the Balanced Scorecard.................................................................184
  Focus on four perspectives.............................................................................184
  Key performance indicators (KPIs).................................................................184
  Estimating relative importance of KPIs..........................................................186
  Performance data collection and monitoring...................................................187
Balanced scorecard rules....................................................................................187
Balanced scorecard infopads..............................................................................188
  Defining rule file syntax for balanced scorecard infopads..............................188

Glossary..............................................................................................................199
Preface

For details, see the following topics:

• About this documentation
• User types
• Information on documentation
• Conventions used in this manual
• Product support contact information

About this documentation

This guide is part of the documentation set for Progress OpenEdge Business Process Server.

User types

Progress OpenEdge Business Process Server is a business process management system that can be used by the following types of users:

<table>
<thead>
<tr>
<th>User type</th>
<th>Responsibilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>User type</td>
<td>Responsibilities</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Manager</td>
<td>Responsible for automating business processes in a particular business domain. Business Process Portal's Management module serves as the primary interface to Business Process Server for the Manager, enabling the manager to monitor, analyze, and control business processes. Also uses the Business Process Modeler for modeling and simulation.</td>
</tr>
<tr>
<td>Application Developer</td>
<td>Responsible for creating customized applications for implementing business processes and developing interfaces associated with tasks. Application developers may work closely with Managers to define the requirements of an application, and determine the business processes.</td>
</tr>
</tbody>
</table>

**Information on documentation**

This documentation includes information for the entire range of Progress OpenEdge Business Process Server users. In the following table, we recommend the guides that are most relevant to each type of user.

<table>
<thead>
<tr>
<th>If you are the …</th>
<th>Read the …</th>
</tr>
</thead>
</table>
For the latest Business Process Server documentation updates, see OpenEdge Product Documentation on PSDN (http://communities.progress.com/pcom/docs/DOC-16074).

**Conventions used in this manual**

This document uses the following conventions and terminology notations.

<table>
<thead>
<tr>
<th>Convention (styles and terms)</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>bold</strong></td>
<td>Indicates titles of command buttons, check boxes, options, lists, dialog boxes, and portal page names.</td>
</tr>
<tr>
<td><strong>file path</strong></td>
<td>Indicates folder paths and file names.</td>
</tr>
</tbody>
</table>
### Convention (styles and terms)

<table>
<thead>
<tr>
<th><strong>Italic</strong></th>
<th>Indicates book titles.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Monospace</strong></td>
<td>Represents code segments or examples.</td>
</tr>
<tr>
<td>**Backward slash **&quot;</td>
<td>Indicates the path in Windows environment. For UNIX environment, replace with forward slash &quot;/&quot;</td>
</tr>
<tr>
<td><strong>OEBPS_HOME or %OEBPS_HOME%</strong></td>
<td>Represents the installation folder of Business Process Server, C:\Progress\OpenEdge\oebpm\server.</td>
</tr>
<tr>
<td><strong>STUDIO_HOME or %STUDIO_HOME%</strong></td>
<td>Represents the installation folder of OpenEdge BPM components, C:\Progress\OpenEdge\oebpm\studio.</td>
</tr>
<tr>
<td><strong>JBOSS_HOME or %JBOSS_HOME%</strong></td>
<td>Represents the installation folder of JBOSS server, C:\Progress\OpenEdge\oebpm\jboss.</td>
</tr>
</tbody>
</table>

### Product support contact information

If the product documentation does not provide a solution to your specific issue, or if you need clarification on the issue, then contact our Product Support team. You can contact the team through the Internet, telephone, or postal mail, as per the details provided in the following table.

**Table 1: Product Support Contact Information**

<table>
<thead>
<tr>
<th>To contact by</th>
<th>Use</th>
</tr>
</thead>
</table>
| **Web site**  | http://progresslink.progress.com/supportlink  
If you are an existing customer, then you can log in to the above site for product support. If you are a first time user, then you need to create an account first. |
| **Telephone** | 1-781-280-4999 for US, Latin America and Canada  
1-781-280-4543 for the Product Support Fax Line |
| **Postal Address** | Progress Software Corporation  
14 Oak Park Drive  
Bedford, MA 01730, USA. |

To enable us to quickly answer your questions, please provide the following information:

- Your name, installation site address and the license key for Business Process Server software.
- Your Business Process Server version and build number.

---

1 For support telephone numbers and offices in your region, visit the support web site above. This contact information is for customer support only.
- Your operating system, application server and browser, with version and service pack details, if any.
- Your database management system and version, and information on JVM and JDBC used.
OpenEdge Business Process Server overview

Progress Software Corporation is a leading global provider of automated business process management solutions. The company's product, Progress OpenEdge Business Process Server (henceforth referred to as Business Process Server or BP Server), is a comprehensive business process management platform, which enables companies to quickly transform their business processes into flexible and manageable Web applications, distributed over intranets, extranets, and the Internet.
Business Process Server addresses every stage in the business life cycle: define, integrate, publish, monitor, analyze, improve, and control. By adopting an end-to-end approach, Business Process Server incorporates all the key elements required to meet the ever-changing demands of e-business while ensuring e-business success. By providing integrated management tools, Business Process Server lets you monitor operations proactively, modifying automated processes dynamically based on changing external operations online. An overview of a typical automated business process management solutions is shown in Figure 1: Business Process Server overview on page 18.

**Figure 1: Business Process Server overview**

For details, see the following topics:

- Business Process Server components
- How Business Process Server works
- Business Process Server user types
Business Process Server components

Business Process Server is a suite of integrated components that enables you to easily build intranet, extranet, and Internet applications and manage your e-business. Business Process Server consists of the following components as in Figure 2: Business Process Server components on page 19:

**Figure 2: Business Process Server components**

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Business Process Portal Home</td>
<td>The Home module of Business Process Portal is the primary interface for application users, enabling them to interact with Business Process Server applications. Users can complete entries to various tasks and applications, and link to the support infrastructure required to complete these tasks.</td>
</tr>
<tr>
<td>Business Process Portal Management</td>
<td>The Management module enables managers to query, report on, and control processes and resources, visible only to the managers.</td>
</tr>
<tr>
<td>Business Process Portal Administration</td>
<td>The Administration module enables Business Process Server Administrators to modify configuration parameters, manage user or group access control, and install or uninstall Business Process Server applications.</td>
</tr>
</tbody>
</table>
### Component Description

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web services</td>
<td>This component allows application developers to: publish their applications as Web services, and find and convert other available Web services on the Internet into Business Process Server applications.</td>
</tr>
<tr>
<td>BPM Webflow</td>
<td>This component provides a framework for developing and implementing Web-enabled workflow applications.</td>
</tr>
<tr>
<td>Progress Developer Studio for OpenEdge</td>
<td>This is the primary application development tool for Business Process Server, which provides an Eclipse-based integrated development environment in which users can create BPM projects, processes, Web applications, and rule files.</td>
</tr>
<tr>
<td>Business Process Modeler</td>
<td>This tool is used to design templates for basic business processes, and to run simulations of processes and individual worksteps.</td>
</tr>
<tr>
<td>BP Server</td>
<td>This is a flexible, lightweight, scalable workflow process engine for intranets, extranets, and the Internet.</td>
</tr>
<tr>
<td>BPM Events</td>
<td>This open, event-driven rule engine is used to formulate and enforce policies in business applications.</td>
</tr>
</tbody>
</table>
How Business Process Server works

The following figure provides an overview of the interaction between Business Process Server components.

Figure 3: How Business Process Server works

The following explanations correspond to the labels shown in the above figure, and describe how the components operate.

1. Progress Developer Studio for OpenEdge and Business Process Modeler provide an integrated development environment (IDE) for Business Process Server, where you can design and publish business processes. The application developer designs a process template (with the *.spt or *.swt extension) in the IDE that reflects the business flow and other business process requirements. Business rules for the process template can be defined using the Rule Editor, a BPM Events component that is launched with Progress Developer Studio for OpenEdge.


3. Once the process template is defined, Business Process Server Administrators use the Administration module to install the business process on the BP Server. Administrators can also configure Business Process Server components, manage user or group access control, and publish Business Process Server applications as Web services. Once installed, users access applications through servlets that pass the requests over an RMI/IIOP connection to the BP Server within an EJB Container.
4. The EJB Container provides a runtime environment that executes and manages Java-based program components that run on the server side of a client/server network. Within the EJB Container are the BP Server and BPM Events server.

5. The BP Server writes events to event tables in the database. Each Business Process uses JDBC to connect to database server as well as store events in the database. Within the BP Server, BPM Process Store uses JDBC to connect to the database server process and retrieve the events deposited by the BP Server process. BPM Process Store interprets the events and populates the process tables. These populated tables are used by Business Process Portal modules.

6. Once the process template is installed as a Business Process Server application, application users can use the Home module to do the following:
   - Access applications
   - Obtain information to perform their tasks
   - Launch the application to start process instances from the BP Server

7. Once the process template is installed as an Business Process Server application, managers can use the Management module (if they have access privileges) to monitor execution of process instances and create reports. Servlets receive requests from managers and pass them onto the BP Server over an RMI/IIOP connection. Managers use the Report Builder to define management reports that retrieve information through JDBC to the database server.

8. BPM Events is a rule-based event or message processing server that loads application rules and executes them against the BP Server and/or external events or messages. This server persists data in the database for recovery and with the help of JDBC connects to the database.

9. Managed Adapters exchange information between Business Process Server applications and external applications by converting Business Process Server-specific protocol to the protocol of an external system such as a database or ERP system. When users add a Managed Adapter to a work step, they can define complex mapping between Business Process Server dataslots and adapter inputs or outputs of the external application. At runtime when the work step is executed, the Managed Adapter sets the adapter inputs and configuration, and maps the outputs to the appropriate output dataslots.

10. BPM Webflow is a run-time component that executes the presentation flows. This component provides a Model, View, Controller (MVC) paradigm for developing presentation flow-based applications and executing them in a Web container.

11. Business Process Servers Web services component allows BP Server applications to be published as Web services.


**Business Process Server user types**

There are four user types within Business Process Server, Application users, Managers, Application developers, Business Process Server administrators.

**User types**

Each Business Process Server user type is defined below:

- **Application users** — Application Users use Business Process Server applications to coordinate specific business tasks with another department within their company, with another company within their organization,
and/or with a business partner in another organization. The Home module in Business Process Portal serves as the primary interface in which Application Users run Business Process Server applications.

- **Managers** — Managers are typically experts in a particular business domain, such as quality assurance or human resources. They might need to work with managers from other groups in automating some of the business procedures that these groups share. The Management module in Business Process Portal serves as the primary Business Process Server interface for business managers to coordinate and integrate business processes, enabling them to exchange information with one another, and to share functionality over such standard communication channels as the Internet or e-mail.

- **Application developers** — Application developers are responsible for analyzing business processes and developing interfaces associated with creating tasks or processes. Application developers are often not domain experts themselves, but work closely with Managers to define business processes and determine the requirements of an application. Application developers use Progress Developer Studio for OpenEdge or Business Process Modeler to define the business process; the resulting process template file is tested, simulated, published, and run as a Business Process Server application.

- **Business Process Server administrators** — Business Process Server administrators are responsible for configuring Business Process Server components, managing user or group profiles and access control, and installing or uninstalling Business Process Server applications. The Administration module in Business Process Portal serves as the primary interface for Business Process Server Administrators to administer applications.

All Business Process Server user types can communicate by using one or more Business Process Server applications. They can also communicate with external applications.
Introducing Business Process Server application development

This chapter outlines the steps for developing a Business Process Server application, and presents sample applications.

For details, see the following topics:

• Outlining Business Process Server application requirements

Outlining Business Process Server application requirements

Before you begin developing a Business Process Server application, you must first collect the requirements for the application from users and managers. Once collected, your next step is to analyze this information to design the existing business processes. Only then can you proceed to translate the user requirements into a process flow and information flow using BPM Designer and Business Process Portal.

Application requirements vary from one application to another. However, most have multiple elements and features in common, as described in the following table.
Table 3: Application development requirements

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Application developer’s tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Designing the process flow of the application</td>
<td>Define the process. Define one or more subprocesses, when applicable.</td>
</tr>
<tr>
<td>Developing the information flow of the application</td>
<td>Define the input and output data required for each workstep. Define the data transfers between the worksteps. Define the information required to execute a workstep.</td>
</tr>
<tr>
<td>Identifying the performer(s) for a specific task</td>
<td>Determine the performer for each workstep type. For example, the performer of an Activity workstep could be an individual, a list of individuals, a group or a queue; for an Adapter workstep, the performer is an external application; for a Subprocess workstep, the performer is a process.</td>
</tr>
<tr>
<td>Defining the look and feel of each Activity workstep</td>
<td>Determine the interface for the workstep In BP Server applications. Interface options include:</td>
</tr>
<tr>
<td></td>
<td>• Auto generated that automatically generates an HTML form in Business Process Portal.</td>
</tr>
<tr>
<td></td>
<td>• Custom that generates a custom JSP form.</td>
</tr>
<tr>
<td></td>
<td>• Flow that starts a Web (formerly BPM Workflow) process from a BP Server process by redirecting the Web process to the BPM Workflow server.</td>
</tr>
<tr>
<td></td>
<td>• Flow (private) that starts a Web process restricted to the current process.</td>
</tr>
<tr>
<td>Controlling the application</td>
<td>Define application control functions, business rules, and management reports. You need access permissions to view and analyze management reports.</td>
</tr>
</tbody>
</table>

While developing applications, you must be familiar with the Business Process Server concepts and terminology. You can refer to Business Process Portal User’s Guide and OpenEdge Getting Started: Developing BPM Applications with Developer Studio for more on Business Process Portal and Designer concepts.

For more information on Business Process Server-related terminology, see the Glossary section at the end of this guide. For a fuller listing of Business Process Server terminology, see the OpenEdge Business Process Server: Terminology Guide.

---

2 BPM Events rules are flexible, capable of monitoring Business Process Server applications, synchronizing process flow, and generating management reports.
Developing an application

This chapter reviews basic application development principles using BPM Designer in Progress Developer Studio for OpenEdge. In BPM Designer, you can create a BPM project or a Web application. For detailed information about each component, see the *OpenEdge Getting Started: Developing BPM Applications with Developer Studio*.

For details, see the following topics:

- Application development stages
- Using a sample application to demonstrate application development
- Using performers in activity worksteps
- Selecting workstep interfaces
- Using the BPM Workflow Web interface generator
- Using BPM Workflow JSP tags
- Determining duration, due date, and overdue periods
- Using JavaScript in a workstep
- Publishing the application
- Running the application
- Removing and reinstalling an application
- Suspending/Resuming installed applications
**Application development stages**

The Business Process Server application development process is summarized as follows:

1. Determine the requirements of the application.
2. Determine the type of application to be created, that is, BPM Project or Web (formerly BPM Workflow). A BPM Project is installed in and run through Business Process Portal. A Web application also acts as a Web-enabled application.
3. Define the application properties, including its name. See *Constraints when naming applications* on page 28.
4. Define the process flow and information flow.
5. Determine the presentation format (interface) of the application's process flow.
6. Verify that the application is complete. Publish and install the application.
7. View and analyze the results of the application. You can also create a simulation project containing the application.

**Constraints when naming applications**

When naming an application, do not use any of the Business Process Server system table names listed below:

- BizEvent
- BizStoreEventCounter
- BizPulseEventCounter
- ProcessTemplate
- ProcessXml
- ProcessDataInfo
- ProcessWorkstepInfo
- ProcessInstance
- SubprocessInfo
- Workstep
- Workitem
- infopad
- infopad_dimension
- infopad_label
- infopad_property
- infopad_slot
- BizLogic_ProcessTemplate
- BizLogic_Sequence
• BizLogic_WorkstepStub
• BizLogic_Workitem
• BizPulse_Sequence
• BizPulse_EventExtent
• BizPulse_InternalEventExtent
• BizPulse_RuleExtent
• BizPulse_DeactivatedRuleExtent
• BizPulse_ScheduledItemExtent
• stat_app_avg_duration
• stat_app_due_info
• stat_app_priority
• stat_app_status_history
• stat_wi_due_info
• UMUser
• UserAttr
• UMGroup
• GroupAttr
• GroupUserRole

In addition, do not use the following two infopad names (created by BPM Events).

• CLASS_INFO_TABLE_INFOPAD
• INFOPAD_ALARM_TABLE_INFOPAD

Also, do not create any applications with the following name, since these tables are created by BP Server at run-time.

• BIZLOGIC_DS_n
• BIZLOGIC_GLOBALDS_n
• BS_n (where n is the serial number)
• <Process template name>

For example, if you install the application 'Hiring', and if its process template ID is 10, then BP Server creates the following tables.

• BIZLOGIC_DS_10
• BIZLOGIC_GLOBALDS_10
• BS_10
• HIRING
Using a sample application to demonstrate application development

This chapter uses the "Assign_A_Task_V1" sample application to illustrate the application development process, beginning with defining the worksteps and dataslots. "Assign_A_Task_V1" is a BPM Project that uses Progress Developer Studio for OpenEdge to define the process and information flow.

The Assign_A_Task_V1 application has the following basic requirements:

- The assigner (a person) assigns a task to the assignee (a person).
- The assignee performs the task.
- The reviewer (a person) reviews the completed task.

In this application, the person assigning a task is also the task reviewer. This person reviews the work completed by the assignee to determine whether it is satisfactory. If it is, then the task reviewer expresses the decision as "NoMoreWork." If there is more work to do, then the reviewer expresses this decision as "DoMoreWork" and can modify the task specification, and reassign it to the current assignee or to a new person.

After defining the application requirements, continue with the application development by transforming the application requirements to a process flow and defining the information required by the application. This includes how the information should flow in the application.

Note: Assign_A_Task_V1 uses the default user interface provided by Business Process Server. Subsequent sample applications are built upon this foundation, and each application is uniquely named, enabling you to run all versions of the application at the same time for comparison.

Defining the process flow of an application

The process flows from "Start" to "End" with the core processing in between. The application has two possible repeating steps:

- **DoTask.** Business Process Portal user assigned to the task, executes the task.
- **ReviewTask.** The manager (or task assigner) reviews the task and decides to either accept the task as completed (Done) or return the task to the assignee for more work (Redo).

**Figure 4: Assign_A_Task_V1 Application Worksteps**
Assign A Task V1 worksteps

Worksteps, or steps in the business process flow, are units of work. In this process template, there are five worksteps. Connectors (lines) connecting the worksteps.

The Assign A Task V1 sample application contains the following worksteps:

Table 4: Assign A Task V1 worksteps

<table>
<thead>
<tr>
<th>Workstep</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>The &quot;Start&quot; workstep is a mandatory initial workstep for every Business Process Server application as it performs the dataslot initialization necessary to properly start the application. For BP Server applications, this initialization is seen in Business Process Portal, when the user opens the application by clicking its link in the Business Process Server Applications page that opens to the Application Details page for the selected application.</td>
</tr>
<tr>
<td>DoTask</td>
<td>This workstep is specific to the application, and corresponds to the actual execution of the task. Assignees see this task in their Task List in Business Process Portal. To complete the task, assignees enter data in the &quot;Comments&quot; dataslot and click Complete.</td>
</tr>
<tr>
<td>ReviewTask</td>
<td>The reviewer reviews the task execution and indicates whether the job has been completed or not by selecting an option from the Decision dataslot in Business Process Portal. In our sample, the reviewer is the same person who started the application, a name which Business Process Server automatically enters in the CREATOR dataslot. As indicated in the workstep’s Properties view, @CREATOR is the Performer, the @ symbol indicating that the performer is dynamically obtained from the CREATOR dataslot at run time.</td>
</tr>
<tr>
<td>Decision</td>
<td>The Decision workstep represents choices the reviewer has made in the previous &quot;ReviewTask&quot; workstep. The reviewer selects &quot;Redo&quot; from the &quot;Decision&quot; dataslot when the task has not been adequately completed and there is more work to do. Using this dataslot value, the Decision then directs the workflow to the &quot;DoTask&quot; workstep through the RequiresMoreWork connector. In the &quot;WorkCompleted&quot; case, the work is completed and the flow continues to the &quot;End&quot; workstep.</td>
</tr>
<tr>
<td>End</td>
<td>This is the final workstep. Some applications might have more than one ending workstep, though our example has only one. Note that there are no particular attributes attached to this step.</td>
</tr>
</tbody>
</table>

Using dataslots to define the information flow

Process flow is only one part of application development; another part is information flow. In designing an application, you must answer the following questions:

- What information do you need for the business process?
- How can you pass information from one step to another?
- At each step, what information requires reading and processing?
Dataslots are used to manage the flow of information in an application. A dataslot is a global variable that helps manage information for a Business Process Server process. The lifespan of a dataslot is the duration of the Business Process Server process instance associated with it. Dataslots are either predefined (system-wide) or user-defined (application-specific). You can also import or export one or more user-defined dataslots using the Import and Export utility in Progress Developer Studio for OpenEdge. For more information, refer to “Exporting and Importing Dataslots” section of the OpenEdge Getting Started: Developing BPM Applications with Developer Studio.

### About predefined dataslots

Predefined system dataslots are common to all BPM Projects. Predefined system dataslots are readable only by worksteps, and are set when the process instance is created or read at run-time. Business Process Server processes can read the values of predefined dataslots, but they cannot change their values. The predefined dataslots are therefore only available as input dataslots (read-only) but not as output (editable) dataslots.

Only the following predefined dataslots are available for processes designed using OpenEdge BPM Designer:

- **AllDataslots.** Contains (at run-time) all the user-defined dataslots, and the associated values for the user-defined dataslots in a Hashmap.
- **Creator.** Contains the creator of the process instance (at run-time).
- **Priority.** Contains the priority (at run-time) of the process. Priority values consist of Low, Medium, High and Critical.
- **ProcessName.** Contains the Business Process Server process name (at run-time). The process name is a string formed by concatenating the process template name with a unique number.
- **StartTime.** Contains the date (at run-time) when the process instance was started.
- **WorkstepName.** Contains the name of the workstep (at run-time) that the process is currently executing.
- **ProcessInstanceId.** Contains the process instance id.
- **ProcessTemplateName.** Contains the process template name of the process instance

For detailed descriptions and examples of use, see the OpenEdge Getting Started: Developing BPM Applications with Developer Studio.

**Note:** Predefined dataslots are not available for Web applications.

### About user-defined dataslots

A user-defined dataslot is a variable defined by the Application Developer during process template development using OpenEdge BPM Designer in Progress Developer Studio for OpenEdge. A user-defined dataslot can be of the type – CHARACTER, LOGICAL, INTEGER, INT64, DECIMAL, Object, DATETIME-TZ, LIST, Map, or Business Object. Each is defined through the OpenEdge BPM Designer interface in Progress Developer Studio for OpenEdge.

**Note:** For detailed information about creating or editing dataslots, see the OpenEdge Getting Started: Developing BPM Applications with Developer Studio.

### Using dataslots in the process template

Specify the initial/default value of a dataslot at definition time. You have the option to set a:
• Static value; that is, a value that is displayed for each instance of the process. For example, entering "Carl" as the value for the Assignee dataslot results in Carl being the Assignee for each process instance; or

• Dynamic value, by entering a dataslot name, preceded by the @ symbol, that displays the value of the designated dataslot at run time. In this case, entering "@Creator" as the value for the Assignee dataslot means that the person who created the instance is the assignee and this value can change from instance to instance and can reflect changes made at run time.

When the dataslot value is set, many other worksteps are free to access this value. In this manner, dataslots provide the information flow across worksteps.

Only the dataslots specified for a workstep appear in the user interface associated with the workstep. You can use the same dataslot as the input (read-only) or output (editable) of several worksteps. Unique cases include:

• A dataslot defined as both input and output of the same workstep. In this case, the value of the dataslot is usually modified in the workstep by the Business Process Portal user.

• A dataslot defined as the output of one workstep and the input of the successor workstep. The value of the dataslot is therefore passed from the first workstep to the successor workstep.

Note that the summation of sizes of all instance dataslots must not exceed the record size. For more details about the record size specific to your database, refer to 

```
OEBPS_HOME\conf\resources\common\DatabaseMapping.xml file.
```

### Configuring length in document and URL dataslot types

The default value for the length of a document's name in a document dataslot or a URL address in a URL dataslot is 256 characters. If the document's name or URL exceeds the default value, then it may be truncated or not displayed correctly. To ensure that lengthy document names or URL addresses are configured correctly, navigate to 

```
OEBPS_HOME\conf\resources\common\DatabaseMapping.xml and, under the database you are using (for example, <data-source name="openedge">), modify the default value of the mapping parameter for Document or URL dataslot to a value that encompasses the expected size of a URL.
```

If you have changed the size of a dataslot or the record size in the database you are using, then you must use 

```
DatabaseMapping.xml to make the corresponding changes in Business Process Server. Business Process Server and your database must be synchronized in this aspect to work correctly.
```

### Defining dataslots in a double byte schema

Business Process Server supports double byte schema. In a double byte schema, most CHARACTER type columns are implemented as NCHAR or NVARCHAR, and such types occupies twice the space of the specified space. For example, NCHAR(10) is 20 bytes in length.

When developing applications in a double byte environment, you should carefully calculate the string length. Business Process Server cannot install applications when the number of bytes for an application table exceeds 8,039. The default size for CHARACTER dataslots is 255. On a double byte schema, the default size is 510. When possible, specify a smaller size in the Size field of the Storage tab in the New Dataslot dialog box.

**Important:** If the multi byte schema is set to True in the DatabaseMapping.xml file, then the byte limitation decreases to half—4030 bytes. If set to False, then the limitation remains at 8,039 bytes.

### About LIST dataslots

The LIST dataslot type is a feature that enables you to create a dataslot that contains a list of choices. This list of values can then be selected by other LIST dataslots from the options provided in the Values panel in the New Dataslot dialog box. Open the Default Format tab to enter a label for the dataslot.
There are several advantages in using a LIST type of dataslot instead of entering CHARACTER dataslot types in the **Choices** dialog box. At run time, the elements of a LIST dataslot can be accessed directly through basic actions on the provided adapters. Furthermore, if the LIST dataslot is mapped to a database table, then the values are read directly from database and no other action is required (as opposed to CHARACTER dataslots mapped to a database, which need concatenation of all the values coming from the table into one long string with separator characters).

Once you declare a dataslot and specify that its possible values (for example, choices) are read from another LIST dataslot, you can specify the type of presentation (for example, drop-down list, radio-buttons, check-boxes) and eventual parameters including width, height, length, and source of choices.

### Creating document dataslots

You can use the Document dataslot to attach a single document or a bundle of multiple documents to an application and upload or download the documents to the server. There are no restrictions on file type. You must specify the Document dataslot contained in a workstep as an editable (output) dataslot in order to attach the file.

**To create a document dataslot:**

1. Open the **Dataslots** tab in the Developer Studio user interface, and click **New** to open the **New Dataslot** dialog box.
2. Type a unique ID for the dataslot in the **Enter a unique ID** field.

   **Note:** The ID cannot exceed 28 characters in length. Valid dataslot IDs must start with a letter. Dataslot IDs can contain letters, numbers and underscores, but cannot contain blank spaces and multibyte characters. While naming a dataslot, be aware of restrictions that can result from certain third-party software limitations.

3. Select the **Document** option from the list of dataslot types, and click **Next**. The **Value** page appears.
4. Select the **Single document** option to attach a single document or **Document bundle** option to attach multiple documents to the dataslot.

   Click the **Browse** button to select documents, and then click **OK** to attach documents in the Document dataslot. You can also enter a URL (http, https or file). In the case of Document bundle, you can add a number of files.

   If you select the **Document bundle** option, then select the **Editable by the author only** checkbox to prevent other users from modifying your document within the dataslot.

   Click **Next**. The **Dataslot** page appears.

5. Enter the description for the Document dataslot, and click **Finish** to complete the creation of the Document dataslot.

   **Note:** For more information on defining or modifying the dataslot format and access, see the *Developing BPM Applications with Developer Studio* manual in the OpenEdge documentation set.

### Document dataslot scenarios

The following table provides various scenarios resulting from selections made in the **New Dataslot** dialog box.
Table 5: Document dataslot scenarios

<table>
<thead>
<tr>
<th>Value - single / bundle</th>
<th>Format - editable / non-editable</th>
<th>Performer - one / all</th>
<th>Scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single</td>
<td>Editable</td>
<td>One</td>
<td>A single document is attached, and the performer can edit it. The dataslot always contains the most recently edited document. For example: An employee writes a proposal and sends it to the Manager. The Manager may edit the proposal and then send it to the Senior Manager. The Senior Manager receives the proposal that is already updated by the Manager, and may edit it further before sending it to the General Manager. At any stage, the document may be returned to any of the earlier performers, who can again edit it. Note that multiple versions of the document are never available, avoiding ambiguity. <strong>Note:</strong> At each workstep, there is only one performer, and that performer can edit the single document. This type of Document dataslot is used when the workflow is simple, a strict version control is in place, and the decision making performer only needs to focus on the most recent document.</td>
</tr>
<tr>
<td>Single</td>
<td>Editable</td>
<td>All</td>
<td>A single document is attached, and all the performers can edit it. The dataslot always contains the most recently edited document - the one last edited by any one of the performers. For example: A Manager initiates a leave plan for his team, with individual team member names. This plan goes to all the members of the team, and each member updates the document with his/her own plan. Any individual’s plan does not depend on or affect another member’s plan, and the sequence of members updating the plan is not important. Since it is a public non-confidential document, a single document can serve the purpose. The Manager can at any time open the document to check whether the plan is complete. This type of Document dataslot is suitable for routine public documents that have specific limited input from all the performers, and the sequence of update is unimportant. Another example is a weekly status report.</td>
</tr>
<tr>
<td>Single</td>
<td>Non-editable</td>
<td>One</td>
<td>A single, read-only document, and the performer can only read the document. Any revision must be conveyed to the creator separately. This type of Document dataslot is appropriate when the specific document is solely owned by the creator, or when it is available to the performer only for information. <strong>Note:</strong> Only one version of the document can exist, and the creator is always responsible for the updates.</td>
</tr>
<tr>
<td>Single</td>
<td>Non-editable</td>
<td>All</td>
<td>A single, read-only document, and all performers can only read the document. Any revision must be conveyed to the creator separately. This type of Document dataslot is used when the specific document is solely owned by the creator. In special cases, the creator may get feedback from performers, but can use discretion to apply the feedback. It can also be conveniently used to communicate information to a large group (for example, company-wide information of a new customer) where no feedback is expected. <strong>Note:</strong> Only one version of the document can exist, and the creator is always responsible for the updates.</td>
</tr>
</tbody>
</table>
### Scenario

<table>
<thead>
<tr>
<th>Value - single / bundle</th>
<th>Format - editable / non-editable</th>
<th>Performer - one/ all</th>
<th>Scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bundle</td>
<td>Editable</td>
<td>One</td>
<td>One or more documents are attached. The performer can edit or remove the documents, as well as attach his/her own documents. The dataslot contains multiple documents that can be attached by different authors, and are of the most recently edited version. For example: An employee writes a proposal and sends it to the Manager. The manager may attach some more documents for reference. The Manager may edit the proposal and reference documents, and may add more supporting documents before sending them all to the Senior Manager. The Senior Manager receives the proposal (already updated by the Manager), and other documents. and may edit one or more documents, attach additional documents, and send them all to the General Manager. At any stage, the documents may be returned to any of the earlier performers, who can again edit them. Note: At each workstep, there is only one performer, and that performer can edit the multiple documents. Note that this situation may result in a loss of prior edits. Note that any of the performers can have the original document, as well as their own edited version of the document attached in this dataslot. Hence multiple versions of the same document may be available at any time. This may be a necessity, when the process requires such tracking.</td>
</tr>
<tr>
<td>Bundle</td>
<td>Editable</td>
<td>One - Editable by the author only</td>
<td>One or more documents are attached. The performer can only read the documents authored by others, and can attach his/her own documents. The dataslot contains multiple documents, that can be attached and edited by different authors, and are of the most recently edited version. Note that each document can be edited only by the performer who attached it to this dataslot. For example: An employee writes a feedback form and sends it to the Manager. The Manager may want some changes, but should not be able to do so without the employee’s knowledge. The Manager may give his review separately, and the feedback form is updated only by the employee. This type of Document dataslot is very useful when the specific documents are solely owned by their creators. Note: Only one version of each of the documents can exist, and their creators are always responsible for the updates.</td>
</tr>
<tr>
<td>Bundle</td>
<td>Editable</td>
<td>All</td>
<td>One or more documents are attached. All the performers can edit all of these documents. The dataslot contains multiple documents that can be attached and edited by different authors, and are of the most recently edited version. Note that each document can be edited by any of the performers, and that this situation may result in a loss of prior edits. For example: A marketing manager initiates a set of proposals, that require inputs from various departments such as finance, development, HR, and administration. This set goes to the managers of these four groups, and each performer updates the set with own inputs. Individual inputs are only for their own department, and the sequence of updates is not important. The marketing manager can at any time open the set to check the latest update. This type of Document dataslot is suitable for routine public documents that have specific limited input from all the performers, and the sequence of updates is unimportant. Another example is an intranet knowledge base with tracking. Note: Any performer can have the original documents, as well as his own edited version of the documents attached in this dataslot. Hence multiple versions of the documents may be available any time. This may be a necessity, when the process requires such tracking.</td>
</tr>
<tr>
<td>Value - single / bundle</td>
<td>Format - editable / non-editable</td>
<td>Performer - one/ all</td>
<td>Scenario</td>
</tr>
<tr>
<td>------------------------</td>
<td>-------------------------------</td>
<td>---------------------</td>
<td>----------</td>
</tr>
<tr>
<td>Bundle</td>
<td>Editable</td>
<td>All - Editable by the author only</td>
<td>One or more documents are attached. The performers can only read the documents authored by others, and can attach their own documents. The dataslot contains multiple documents that can be attached by different authors. Note that each document can be edited only by the performer who attached it to this dataslot. For example: A procedures dataslot may contain multiple procedures, however, each one is owned by a specific person. If any changes are required, only the creator can carry out the changes. This type of Document dataslot is very useful when different documents are owned by different performers, but are available for public viewing. <strong>Note:</strong> Only one version of each of the documents can exist, and their creators are always responsible for the updates.</td>
</tr>
<tr>
<td>Bundle</td>
<td>Non-editable</td>
<td>One</td>
<td>Read-only documents. The performer can only read the documents. Any review must be conveyed to the creator separately. This type of Document dataslot is appropriate when the specific documents are solely owned by the creators, or when they are available to the performer only for information. <strong>Note:</strong> Only one version of each of the documents can exist, and their creators are always responsible for the updates.</td>
</tr>
<tr>
<td>Bundle</td>
<td>Non-editable</td>
<td>All</td>
<td>Read-only documents. The performers can only read the documents. Any reviews must be conveyed to their creator separately. This type of Document dataslot is used when the specific documents are solely owned by their creators. In special cases, the creators may get feedback from performers, but can use their discretion to apply the feedback. It can also be conveniently used to communicate information to a large group (for example, organization policies) where no feedback is expected. <strong>Note:</strong> Only one version of each of the documents can exist, and their creators are always responsible for the updates.</td>
</tr>
</tbody>
</table>

**Using performers in activity worksteps**

For each Activity workstep in Assign_A_Task_V1, you can assign a performer in the Performer box in the **General** tab of the **Properties** view. In the DoTask workstep, the performer is "Value of ‘Assignee’", that is, the initial performer is linked to the Assignee dataslot. You can dynamically change the initialized performer by changing the value in the Assignee dataslot.

1. In the **Performer** dialog box, click **Modify** to open the **Performer** box.
2. Click the **ellipse** button beside the **Name** box and then the **Select a Dataslot** option to display the **Select a Dataslot** dialog box.
3. Select the dataslot and then click **OK** to modify the performer.

In the ReviewTask workstep, the Performer is shown as "Value of ‘Creator’". Here, the performer is specified as a variable in the process definition, the variable being the dataslot name preceded by the character '@'. The performer is the run-time value of the dataslot when the task is performed.

**Note:** For additional information about setting the Performer values in application development, see the *OpenEdge Getting Started: Developing BPM Applications with Developer Studio*. 

---

**OpenEdge Business Process Server: Application Developer's Guide**  37
Selecting workstep interfaces

Business Process Server supports the following user interfaces:

- **Portal** for a browser
  - **Auto Generated**: Provides a default HTML presentation, automatically generated at runtime.
  - **Custom**: Generates custom JSP forms.
  - **Flow and Flow (private) interfaces**: Provides both Flow and Private Flow interfaces.

- **Mobile** for a Mobile device
- **Tablet** for a Tablet device
- **GUI** for an OpenEdge GUI for .NET application
- **Other**

**Note:** For more information on workstep interface options, see *Defining Workstep Presentation format* in the *OpenEdge Getting Started: Developing BPM Applications with Developer Studio*.

The default HTML form does not facilitate the following:

- Validation of input
- Generation of meaningful error messages
- Control of the interface design
- Addition of images, such as company logos
- More descriptive titles and headings
- Different layout of fields or the reorganization of data fields

Using the BPM Workflow Web interface generator

The BPM Workflow Web Interface generator uses the data in a Web application to generate a customized interface (or presentation page) in your Web browser. The generator is provided as a set of Java interface classes, with implementations for a range of presentation technologies that include JSP, XML and XSL. Extensions to support additional presentation technologies can easily be supported by plugging in their corresponding implementations.
JSPs generated for a default presentation

Business Process Server provides an implementation of an XSL transformation file—generator.xsl—in the OEBPS_HOME\BPMWebFlow\common\xsl directory. This transformation generates JSP code and operates directly on the XML process definition for the requested application. Additional input, such as the name of the page to be generated, miscellaneous configuration parameters required during transformation and any other data which might be required, is provided as input to the transformation through the use of transformer parameters.

The result of the generator.xsl transformation is JSP pages. The transformation is applied repeatedly for each Activity workstep defined in the process template for a Web application in BPM Designer. Logic code, such as the controller page for a Web application or the logic section of a Business Process Portal JSP page, are still generated by a Java generator. If requested, the generation of the JSP logic uses a separate transformation as well.

By default, Business Process Server provides only one default transformation for both Business Process Portal JSP pages and Web application JSP pages. However, if you, as a user or application developer, want to create a different default transformation for an application, then make a copy of the existing generator.xsl in the OEBPS_HOME\BPMWebFlow\common\xsl/ folder, make changes on the copy of the default XSL transformation file, and place it under a directory that you have created to contain it; for instance, OEBPS_HOME\BPMWebFlow\<appName>\generator.xsl. By default, BPMWorkflow first looks for the generator.xsl file in the Application folder. If it does not find this transformation file there, then it looks for the default transformation file, located in the OEBPS_HOME\BPMWebFlow\common\xsl directory.

Using BPM Workflow JSP tags

At run time, BPM Workflow facilitates the generation of JSP-based interfaces by using XML-like tags and scripts to generate an interface page. These tags are maintained in the JSP Tag Library, allowing developers to introduce additional complexity into their JSP pages without discarding the general "look and feel" of the JSP pages or the user’s ability to customize these types of pages. The following table presents a list of the JSP tags defined in BPMWebFlow.

Attributes in bold and italic (example, name) are required fields. The default in a group of options (example, STRING) is underlined.
### Table 6: Defined JSP 1.1 tags

<table>
<thead>
<tr>
<th>Name</th>
<th>Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bizsolo:initApp</td>
<td><code>&lt;name=&quot;name&quot; mode=&quot;mode&quot;/&gt;</code></td>
<td>Creates context at application’s initialization. Includes the generateID functionality, which creates a String representation of a unique ID and makes it available to the page context, under the specified name.</td>
</tr>
</tbody>
</table>
| bizsolo:initDS | `<name="STRING, LONG, DOUBLE, ..." type="TYPE" label="label" value="value" param="param" hexval="TRUE|FALSE" choices="choices"/>
|                |                                                                                     | Creates a new BPMWebFlow dataslot, with the given name, type, value and choices. If the type is not specified, then the default type is CHARACTER. Allowed values for type are: CHARACTER, INTEGER, INT64, DECIMAL, BOOLEAN, OBJECT, LIST, DATETIME-TZ, RADIO, CHECKBOX, SELECT, MSELECT, and TEXTAREA. The label for the dataslot can be specified using label = "label". The value used for initialization can be specified directly, using value="value", or it can be taken from the HTTP request object, by searching for a request parameter with the same name. If the parameter name is different, then it can be specified by using param="param". If the value, either specified directly or taken from the request object, is hexencoded, then this has to be specified by setting hexval="TRUE". |
| bizsolo:setDS  | `<name="name" type="TYPE" value="value" hexval="TRUE|FALSE"/>
|                |                                                                                     | Updates the named BPMWebFlow dataslot. If the value is specified as hexencoded (default is False), then hexencoding is performed prior to updating. If value is set, then the update is performed by using "value" instead of the value retrieved from the Request object. |
| bizsolo:getDS  | `<name="name" type="TYPE" writePerm="TRUE|FALSE" required="TRUE|FALSE" password="TRUE|FALSE" size="40" maxlength="100" rows="5" cols="40"
<p>|                |                                                                                     | Gets HTML interface for named dataslot. If writePerm=&quot;TRUE&quot;, then this dataslot is presented as output, that is, it may return a value as part of the request. Precision and scale parameters are used to define numerical data in DECIMAL type dataslots. |
| bizsolo:executeAction |                                                                                   |                                                                                                                                                                                                             |</p>
<table>
<thead>
<tr>
<th>Name</th>
<th>Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bizsolo:invokeWS</td>
<td><code>&lt;epClassName=&quot;...&quot; perfMethod=&quot;...&quot; [serviceName=&quot;...&quot; wsdl=&quot;...&quot; operation=&quot;...&quot; [port=&quot;...&quot;]]]</code></td>
<td>Invokes a Web service, with a JSP form.</td>
</tr>
<tr>
<td>bizsolo:OpParam</td>
<td><code>&lt;name=&quot;...&quot; value=&quot;...&quot; [msg=&quot;...&quot; [part=&quot;...&quot; [type=&quot;...&quot;]]]]</code></td>
<td>Provides a Web service operation parameter.</td>
</tr>
<tr>
<td>bizsolo:redirectURL</td>
<td><code>&lt; [page=&quot;page&quot;] [application=&quot;application&quot;/&gt;&gt;</code></td>
<td>Redirects to the specified page or application and insulates the user from any details of session handling (URL rewriting)</td>
</tr>
<tr>
<td>bizsolo:transferDS</td>
<td><code>&lt;bizsolo:transferDS/&gt;</code></td>
<td>Used in an End workstep to transfer dataslots back to the BP Server.</td>
</tr>
<tr>
<td>bizsolo:getLabel</td>
<td>`&lt; name=&quot;...&quot; type= &quot;DATASLOT</td>
<td>LINK</td>
</tr>
<tr>
<td>bizsolo:pageInstruction</td>
<td><code>&lt;[value=&quot;...&quot;/&gt;&gt;</code></td>
<td>Creates a tag that handles instructions for worksteps from BPM Designer.</td>
</tr>
<tr>
<td>bizsolo:getHttpPath</td>
<td><code>&lt;id=&quot;id&quot; pageName=&quot;...&quot; appName=&quot;...&quot;/&gt;</code></td>
<td>Returns the HTTP URL of a given page for a particular application. Use it to allow dynamic choices of CSS pages, images or *.conf files used when generating an application. Normally, it is not required if you plan to have only static references in the JSP source.</td>
</tr>
</tbody>
</table>

`<epClassName="..." [dsClassName=["..."]] perfMethod="..." [query="..." [filter="..." [ds="..." [dso="..." [mode="WebApp"]]]]]> [res="..."[/>]

dClassName is made obsolete by className in DsPrototypeFactory. The Mode parameter sets the type of exception report. If you select '"', then exception is reported; if WEBAPP, then no exception is reported within the JSP page.
<table>
<thead>
<tr>
<th>Name</th>
<th>Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bizsolo:getBSParameter</td>
<td><code>&lt;name=&quot;param_name&quot;/&gt;</code></td>
<td>Reads the value of a BPMWebFlow parameter (specified in bpmwebflow.conf) during run-time.</td>
</tr>
</tbody>
</table>
| bizsolo:setCallerData | `[parentApp="parentApp"]
childApp="..." [dsList="..."] [returnPage="..."]`/ > | Stores parent process data in the HTTP session.                           |
| bizsolo:getCallerData | `[parentApp="parentApp"]
childApp="..." [dsList="..."]/ > ` | Retrieves parent process data from the HTTP session.                     |
| bizsolo:setParentData | `<dsList="...">`                                                        | Tells subprocess to retrieve parent process data in the HTTP session.       |
| bizsolo:getParentData | `<dsList="...">`                                                        | Sends retrieved subprocess data back to parent process.                    |
| bizsolo:checkPermission | `<application="...
component="...
permission="...">`/ > | Provides user authorization for a Web application. Establishes access to a specific component, application or user permission. |
| bizsolo:choose       | `<bizsolo:choose/>`                                                       | Executes only one JSP section out of available multiple options. Equivalent to a switch statement. |
| bizsolo:when         | `<est="...">`                                                           | Identifies a JSP section (tag body) to be executed when test condition (run time condition) is evaluated as True. Used only within a `<bizsolo:choose/>` tag. Equivalent to a case statement. |
| bizsolo:otherwise    | `<bizsolo:otherwise/>`                                                   | This default is executed when none of the preceding four tags are evaluated as True. Used only within a `<bizsolo:choose/>` tag. |
| bizsolo:if           | `<test="...">`                                                          | Executes the JSP section (tag body) when test condition (run time condition) is evaluated as True. |
| bizsolo;ifCrtWS      | `<name="...
[param="param"] [isDefault="...">`/ > | Executes the JSP section (tag body) when test condition matches the current workstep name. Used only within the controller page. |
| bizsolo:value        | `<name="...
index="0">`                                                 | Returns the value of the specified dataslot. If the dataslot has multiple values, for example, combo box, then use the index attribute. |
### Using the `getLabel` tag

The `getLabel` tag is used to return a label. Users can associate the label with the following types: a dataslot (in a specified workstep), a link (connection), or a resource (system or user-defined). If `mode=WebApp`, then the language (Locale) is set by the user browser preferences that are sent as part of the HTTP request. The default mode is `WebApp`, and we recommend that it is not modified during JSP customization.

The `type` attribute can have one of the following values: `DATASLOT`, `LINK`, and `RESOURCE`. If `type=DATASLOT`, then you must also use `wsName` (workstep name) attribute. If `type=RESOURCE and resClass` is not defined (the default), then internal resources are returned. If `type=RESOURCE and resClass` is defined, then user-defined resources can be returned. The algorithm used is the same as for any Java ResourceBundle. The resources (Java classes or property files, specified by the tag parameter "name") must be available for loading by either the appserver classloader or the Business Process Server classloader, which is the preferred method for packaging reasons. The classloader currently used by Business Process Server does not support reloading of adapter classes that have been updated unless the servlet engine is restarted.

### Determining duration, due date, and overdue periods

In Business Process Server, you typically specify the duration of a process or a workstep when you create the process or add the workstep to a process template. Duration is defined as the period of time you anticipate it takes to complete the process or workstep. Due date is calculated by adding the duration of the process or workstep to the time the process or workstep is actually activated (It is also possible, however, for the client to set a specific due date by invoking the appropriate BP Server APIs). When the duration is not specified, the due date of an activated workstep is set to 'activation time of Workstep + 2 hours' and due date of an activated process is set to 'activation time of process + 1 day'. The Overdue period starts when the due date is passed.
To determine the period of time that a process instance or workstep instance has been running:

• For a process instance, inorder to obtain the period of time that a process instance has been running, read the value of start time by calling getStartTime() API against the 'ProcessInstance' object and compare it to the current system time.

• For a workstep instance, obtain the start time by calling getStartTime() API against the WorkStepInstance object and compare it to the current system time.

To determine the period of time remaining before a process instance or workstep instance becomes overdue, obtain the due date through the getDueDate() API against the ProcessInstance or WorkStepInstance objects and compare it to the current system time.

**Setting overdue actions in a workstep**

In many business scenarios, applications are required to take specific actions when a task is not completed by the defined due date. Business Process Server enables users to select one or more options from a list of predefined tasks/actions that can be directly associated with the overdue worksteps.

By default, the Overdue Action daemon activates every 30 seconds and starts to check all the worksteps that are activated, have due dates, and have timer actions specified for overdue. This duration is set in the file OEBPS_HOME\conf\bpserver.conf for the parameter bizlogic.timer.schedule.checkdue. The default value is 30 seconds, and the recommended minimum level is ten seconds. If the duration is set to zero or less than zero, then the default value of 30 seconds is used. For more information on setting duration for overdue actions, see Action details for overdue and last overdue tasks on page 46.

**Note:** Changing the bizlogic.timer.schedule.checkdue value may affect the performance of BP Server.

When the overdue date or time is reached, the overdue actions you specified are automatically executed. The overdue actions are executed for the specific number of times the workstep has reached the due date. The overdue tasks are executed each time the due date is reached until one of the following conditions are satisfied:

• The maximum number of repetitions is reached.

• The workstep is completed.

• The process instance is completed.

**To set overdue actions:**
1. Click the On Overdue tab of the Advanced tab of the workstep’s Properties view.

   Figure 5: The On Overdue tab of the Advanced tab of the properties view

2. To add an overdue action, click New to open the Overdue Actions dialog box, which contains three tabs: Actions, where you can define the actions that take place when the workstep is overdue; Script, where you can enter JavaScript to perform overdue actions; and Alerts, where you can add alerts to be triggered on overdue.

   Figure 6: Overdue actions dialog box

3. Type the number of times you want the action repeated in the Number of times to execute box.

   If the Number of times to execute field is set to a value=2 for action A1, then the action is repeated twice. If a particular execution of the action fails, then the execution is retried again.

4. Select one or more of the checkboxes to set the Overdue actions. For more information on adding details, refer to “For an Overdue Workstep” section of the OpenEdge Getting Started: Developing BPM Applications with Developer Studio.

   Table 7: Action details for overdue and last overdue tasks on page 46 lists the Overdue actions in detail.
5. After defining the Overdue actions, click **OK**, returning to the **On Overdue** tab of the workstep’s **Properties** view. The Overdue actions you have defined are now displayed in the Overdue actions table.

6. You can also set an action in the **On last overdue** section (Figure 5: The **On Overdue** tab of the Advanced tab of the properties view on page 45) of the **On Overdue** tab of the workstep’s **Properties** view. This action is executed only on the final overdue of the workstep, and is only repeated once, and contains no other actions.

### Action details for overdue and last overdue tasks

The following table lists the Overdue Actions and On Last Overdue actions in detail.

#### Table 7: Action details for overdue and last overdue tasks

<table>
<thead>
<tr>
<th>Action</th>
<th>Parameters</th>
<th>Workstep Type</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>For overdue actions</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Send E-mail</td>
<td>&lt;email-ids&gt;&lt;subject&gt;&lt;content&gt;&lt;charset&gt;</td>
<td>Activity, Adapter, and Subprocess</td>
<td>E-mail ID may also be specified as @DS1 to dynamically get the mapped datalot value. Specify multiple target email-ids by using comma separated values. Using &lt;charset&gt;, it is possible to specify the content for default or specific character set; &lt;content&gt; and &lt;charset&gt; are optional.</td>
</tr>
<tr>
<td>Change Priority</td>
<td>&lt;pri&gt;&lt;INCREASE&gt;&lt;DECREASE&gt;</td>
<td>Activity, Adapter, and Subprocess</td>
<td>Priority can be set to a specific value or specified as increase/decrease. The increase or decrease is executed based on the numerical value of the priority.</td>
</tr>
<tr>
<td>Extend Duration</td>
<td>&lt;hrs&gt;&lt;mins&gt;&lt;secs&gt;</td>
<td>All Activities, Adapter, and Subprocess worksteps with defined duration.</td>
<td>When the workstep duration is less than the configured daemon time, it may be possible that, for the “Extend Duration” action, the new due date is already past when the action is executed. In this case, the workstep due date is set to the last minute of the current day (that is, 11:59:00 PM). With a nested workstep, the new due date must also be updated to the actual subprocess instance.</td>
</tr>
<tr>
<td>Execute JavaScript</td>
<td>&lt;js&gt;</td>
<td>Activity, Adapter, and Subprocess</td>
<td>Open the Script tab to enter JavaScript that is executed at each overdue.</td>
</tr>
</tbody>
</table>

**For last overdue actions**
<table>
<thead>
<tr>
<th>Action</th>
<th>Parameters</th>
<th>Workstep Type</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Change Performer</td>
<td>&lt;performer&gt;&lt;mappedDS&gt;</td>
<td>Activity and Flow</td>
<td>Reassign or make workitems available to a user that is hardcoded or a user that is mapped to a dataslot, based on the value shown in Change performer text box. Also supports JS and EJB performers. If the performer of the workstep is Single User, on overdue, then it is reassigned to another Single User. Note: Group (Any/All) not supported in Change Performer action. If the performer of the workstep is All members of Group, then on overdue it is reassigned to another Single User. Note: Group (Any/All) not supported in Change Performer action. For example, on overdue, task assigned to each group member is now assigned to the single user. Therefore, if the Group had say 3 members, then on overdue, the single user has 3 tasks in the list of assigned tasks. If the performer of the workstep is Any member of a Group, then on overdue it can be reassigned to a Single User or a Group (Any/All). Irrespective of the performer type, on overdue, the task is moved to the list of the new performer's &quot;Available tasks&quot;. For example, if the original performer is &quot;Any member of group A&quot;, then on overdue, the performer is changed to ebms, the task is present in the Available task list of ebms.</td>
</tr>
<tr>
<td>Complete workstep</td>
<td>NA</td>
<td>Activity and Subprocess</td>
<td>With Adapter workstep, force complete is supported only for asynchronous adapters (ASYNC EP).</td>
</tr>
</tbody>
</table>

**Using JavaScript in a workstep**

**Note:** Interfaces using JavaScript are developed through OpenEdge BPM Designer only.

For worksteps, you can create customized business logic using JavaScript. Enter and modify JavaScript code in BPM Designer under the Advanced tab in the workstep Properties view. Here you can define logic for a workstep before activation, upon activation, when completed, when overdue, upon an error and upon recovery, as desired. For more information about using BPM Designer, refer to OpenEdge Getting Started: Developing BPM Applications with Developer Studio.
Business Process Server executes the scripts you specify in the Before activation section at the start of the workstep, but after executing the precondition. The scripts you specify in the When completed section are executed at the completion of the workstep. You can also use the AutoComplete feature to automatically add scripts in these two sections. For more information, refer to "Adding Script to a Workstep" section of the OpenEdge Getting Started: Developing BPM Applications with Developer Studio.

The compensatory script is executed if that workstep is involved in a rollback process. For more information about using compensatory script in an application, see Writing compensatory JavaScript code (optional) on page 113.

Using JavaScript in a workstep, you can:

- Manipulate dataslots in a workstep without having to write an adapter.
- Send an e-mail message in any workstep without creating a separate workstep.
- Use JavaScript’s regular expressions, pattern matching, and searching capabilities, in addition to using the Math library to perform calculations on dataslot values.

When you save the process template, it compiles the JavaScript code and rejects all code for which the compilation failed. All JavaScript code associated with the process template is then compiled again using the maximum optimization level when the process template is created in the BP Server. This compiled object is saved with the template workstep for faster execution of the JavaScript. If, however, any of the JavaScript compilation failed, then the creation of the process template is aborted and an exception is sent to the client.

Integrating JavaScript

Each time Business Process Server executes JavaScript code for a workstep, it creates a new object of class JSTools—referred to as jst—and passes it to the JavaScript execution environment. This operation is transparent to you although you must use jst objects to invoke methods for accessing dataslot values, sending e-mail, and more.

This section explains how to integrate your JavaScript code into a workstep.

Reading dataslot values

You can read dataslot values in a workstep using the get() methods. Each of these methods are type specific and you must invoke the appropriate method based on the dataslot type.

The methods for reading dataslot values are:

- Object.jst.getDataSlotValue(String dsName);

For example, you could read the string dataslot empName using the following code:

```javascript
var name = jst.getDataSlotValue("empName");
```

Similarly, to read the double dataslot empSalary, use the following code:

```javascript
var salary = jst.getDataSlotValue("empSalary");
```

Using new datatypes in JavaScript

The following examples show the use of new datatypes in JavaScript.
Using DateTimeTZ in JavaScript

```javascript
var jdt = (new java.util.Date()).getTime();
var currentTime = new Packages.com.savvion.sbm.bizlogic.server.svo.DateTime.getInstance(jdt);
jst.putDataSlot("dsDate", currentTime);
var date = jst.getDataSlotValue("dsDate");
jst.writeLog("Date value <" + date.getStringValue() + ">");
```

Using decimal value in JavaScript

To set decimal values, use the setScale (int scale, int roundingMode).

```javascript
var jdec = new java.math.BigDecimal(2222.555);
var bigdec = new Packages.com.savvion.sbm.bizlogic.server.svo.Decimal.getInstance(
    jdec.setScale(5, java.math.BigDecimal.ROUND_UP));
jst.putDataSlot("dsDecimal", bigdec);
var decimal = jst.getDataSlotValue("dsDecimal");
jst.writeLog("Decimal value <" + decimal.getStringValue() + ">");
```

Using decimal in precondition JavaScript

```javascript
js:function checkDecimal()
{ 
var decimal = jst.getDataSlotValue("dsDecimal");
if (decimal.getValue().doubleValue() > 10.8) {
    return true;
}return false;
}
checkDecimal()
```

Updating dataslot values

Unlike the methods you use to read a dataslot, the only method to update a dataslot is typeless. To update a dataslot, use the following code:

```javascript
jst.putDataSlot(dsName, JavaScriptVariableName);
```

where `dsName` is the actual dataslot name and `JavaScriptVariableName` is the name of the JavaScript variable that holds the new value you want to assign to the dataslot.

**Important:** The datatype of the JavaScript variable should match the datatype of the dataslot.

For example, you could use the following code to read and update a dataslot (without changing its value):

```javascript
var name = jst.getDataSlotValue("empName");
var salary = jst.getDataSlotValue("empSalary");
jst.putDataSlot("empName", name);
jst.putDataSlot("empSalary", salary);
```

If you want to update the `date` dataslot, then the following method provides an example:

```javascript
jst.putDataSlot(String dsName, Object dsValue);
var jdt = (new java.util.Date()).getTime();
var currentTime = new Packages.com.savvion.sbm.bizlogic.server.svo.DateTime.getInstance(jdt);
jst.putDataSlot("dsDate", currentTime);
```
If you want to set decimal values dataslots, then the following method provides an example:

```java
/*
 * For setting Decimal values.setScale(int scale, int roundingMode) should be used.
 */
var jdec = new java.math.BigDecimal(2222.555);
var bigdec = new Packages.com.savvion.sbm.bizlogic.server.svo.Decimal.getInstance(jdec.setScale(5, java.math.BigDecimal.ROUND_UP));
jst.putDataSlot("dsDecimal", bigdec);
```

### Sending e-mail using JavaScript

You can send an e-mail message using JavaScript without having to create a specific workstep just for this task. To send an e-mail message, use the following code:

```javascript
jst.sendEmail(address, subject, content);
```

where `address` is the e-mail address of the recipient, `subject` is the message subject, and `content` is the body of the message.

You can send the values of multiple dataslots by concatenating the dataslot values before sending the message. For example, you could use the following code to send an e-mail message:

```javascript
var jcont = name + ":" + salary;
jst.sendEmail("sbm@savvion.com", "testing", jcont);
```

or

```javascript
jst.sendEmail("sbm@savvion.com", "testing", name + ":" + salary);
```

### Setting e-mail to a different language with JavaScript

You may adjust the character set to a language of your choice for e-mails. This is done with either pre- or post-script. The following example demonstrates sending an e-mail in Japanese:

```javascript
var obj = jst.getSendEmailObject("ISO-2022-JP");
obj.setMailFrom("home@savvion.com");
obj.setMailTo("john@savvion.com");
obj.addMailTo("mary@savvion.com");
obj.setSubject("<Japanese subject>");
obj.appendText("<Japanese message: line 1>");
obj.appendText("<Japanese message: line 2>...");
obj.send();
```

### Invoking Java methods in JavaScript

You can invoke Java methods from the workstep using JavaScript. You might want to do this to perform complex calculations, perform sophisticated business analysis, or to leverage certain functionality within an existing tool. To invoke a static Java method from JavaScript, use the following syntax:

```javascript
Packages.<fully_qualified_java_class_name>.<static_method_name>
```

To instantiate a new Java object, use the following syntax:

```javascript
var obj = new Packages.<fully_qualified_class_name>(params);
```
To instantiate a new array of a Java class, use the following syntax:

```
var dsNames =
java.lang.reflect.Array.newInstance(<fully_qualified_class_name>, <array length>);
```

For example, to create an instance of a string array of size 3, use the following code:

```
var dsNames =
java.lang.reflect.Array.newInstance(java.lang.String, 3);
```

The JavaScript Engine uses the keyword Packages to identify the Java classes and methods.

**Note:** The keyword Packages is not required for classes in Sun Java's JDK, in `classes.zip`, or in `rt.jar`.

The following is an example of how to invoke Java methods in JavaScript:

Java program: Employee.java
```
package com.tdiinc;
public class Employee {
   protected static String companyName;
   protected String empName;
   Employee(String ename) {
      empName = ename;
   }
   public static String getcompanyName() {
      return(companyName);
   }
   public String getempName() {
      return empName;
   }
}
```

JavaScript:
```
function f1() {
   var cmpname=Packages.com.tdiinc.Employee.getcompanyName();
   var empObj = new Packages.com.tdiinc.Employee("George")
   var ename = empObj.getempName();
}
```

**Invoking an adapter for communicating to BP Server API**

To invoke an adapter that talks to the BP Server API using JavaScript, complete the following:

```
function f1(){
   var cls = jst.loadClass(adaptername, path);
   var adpt = cls.newInstance();
   adpt.methodname();
}
```

In the above example, `adaptername` is the java adapter that talks to BP Server and `path` is the absolute directory path.

The next line, `var cls = jst.loadClass;` creates an instance of Class object. Once the Class object is obtained, you can retrieve an instance of the adapter class by calling:

```
var adpt = cls.newInstance();
```

You can invoke methods using the object `adpt` of the BP Server adapter class. Locate the adapter class in the `OEBPS_HOME\ebmsapps` directory.
JavaScripts, however, cannot invoke BP Server adapters. The solution is for the EJBClassLoader to load the JavaScript classes. The JavaScripts function invokes the adapter which calls BP Server using the following code, similar to the above:

```javascript
<PreFunction
  function fl(){
    var cls = jst.loadClass("readOnly");
    var ro = cls.newInstance();
    ro.read();
    fl()
  }
</PreFunction>
```

In the above prefuction, the class object of readOnly (This is the adapter that talks to BP Server) is called by:

```javascript
var cls = jst.loadClass("readOnly");
```

Once the class is obtained, the new instance is called by:

```javascript
var ro = cls.newInstance
```

Now you can invoke methods on the instances.

### Using 'blserver' in JavaScripts

In Business Process Server applications, you can use JavaScripts to provide customized business logic for any workstep. You must frequently access the BP Server APIs to complete the business logic in the JavaScripts. For this, Business Process Server provides an inbuilt object 'blserver' that can access all APIs of the BP Server. Communication to the BP Server in the JavaScript code should start with a call to connect as:

```javascript
var session = blserver.connect(<user>, <password>)
```

Ensure that you end the communication with a call to disconnect from the server as:

```javascript
blserver.disConnect(session)
```

After a connection is established with the BP Server, all features exposed by the BP Server API are available for you to perform various tasks, such as:

1. Working with BP Server specific information such as version, or start time. For example:
   a) BP Server version

   ```javascript
   var session = blserver.connect(<user>, <password>)
   var session1 = blserver.getVersion(session)
   blserver.disConnect(session)
   ```

   b) BP Server start time

   ```javascript
   var session = blserver.connect(<user>, <password>)
   var startTime = blserver.getStartTime(session)
   blserver.disConnect(session)
   ```

   c) BP Server information

   ```javascript
   var session = blserver.connect(<user>, <password>)
   var info = blserver.getInfo(session)
   blserver.disConnect(session)
   ```

2. Working with smart value objects (SVO) for the process template, process instance, workstep template, workstep instance, work item, and dataslot. For example:
a) Setting the process instance creator.

```javascript
var session = blserver.connect(<user>, <password>)
var piName = jst.getProcessInstanceName()
var pi = blserver.getProcessInstance(session, piName)
pi.setCreator(<creatorname>)
pi.save()
blserver.disConnect(session)
```

b) Setting the due date of workstep instances.

```javascript
var session = blserver.connect(<user>, <password>)
var piName = jst.getProcessInstanceName()
var wsName = jst.getWorkstepName()
var wsi = blserver.getWorkStepInstance(session, piName, wsName)
var duedate = new Packages.com.savvion.sbm.bizlogic.server.svo.DateTime
(<date-string>)
wsi.setDueDate(duedate)
wsi.save()
blserver.disConnect(session)
```

**Reading process information**

You can invoke the following methods to read the current workstep name, process instance name, and process template name.

```javascript
jst.getWorkstepName();
jst.getProcessInstanceName();
jst.getProcessTemplateName();
```

For example, to read the current workstep name, use the following code:

```javascript
var wsn = jst.getWorkstepName();
```

**Including a driver function**

You can define as many JavaScript functions as you need for the Before Activation, When Completed, or On Recovery sections of any workstep. However, you must define one driver function.

For example, f3 would serve as the driver function in the following code, and is the one that would be invoked.

```javascript
function f1() {
    . . .;
}function f2() {
    . . .;
    var x=f1(); return(100)
}function f3() { /* driver function */
    . . .;
    var z = f2();
    . . .;
} f3()
```

**Note:** If the driver function returns a value to the caller (which happens to be the BP Server), then this value is ignored. However, all other JavaScript functions may return a value, as required.
Understanding exceptions

Exceptions can occur in the JavaScript code or in one of the Java methods invoked in the script. Whenever an exception occurs during the execution of the JavaScript code, the exception is thrown to the caller, which in this case is the BP Server. The BP Server catches the exception from the JavaScript execution environment, writes the stack and exception messages to the \OEBPS_HOME\logs\bpserver.log file, and suspends the workstep.

Managing collections of commonly used JavaScript code

You might find yourself using certain JavaScript functions repeatedly in a number of different process templates. To avoid duplicating the definition and compilation of this JavaScript code in multiple processes/worksteps, you can put these commonly-used code elements in a JavaScript file called common.js in the \OEBPS_HOME\BPServer directory.

You may define a function and place it in the common.js file. If this is the first function created, then you must create the common.js file first. All JavaScript functions defined in \OEBPS_HOME\BPServer\common.js are compiled during the start-up of the BP Server and are ready for use from any JavaScript code.

Important: Since the JavaScript defined in the common.js file is kept in memory until the BP Server is shut down, you should only add those JavaScript functions that is used by the maximum number of the applications, processes, and worksteps.

For example, if most of your applications access CHARACTER dataslots for URLs, and you want to provide a JavaScript function to download the contents and search for specific data, then you can add the following script to the common.js file:

```javascript
function getURLContent() {
    var url = "http://www.savvion.com";
    var contents = "";
    var nextLine = "";
    var urlObj = new java.net.URL(url);
    if (urlObj == null) {
        return "";
    }
    var ustream = new java.io.DataInputStream(urlObj.openStream());
    if (ustream == null) {
        return "";
    }
    while (true) {
        nextLine = ustream.readLine();
        if (nextLine == null) {
            break;
        }
        contents += nextLine + "\n";
    }
    ustream.close();
    return contents;
} getURLContent()
```

Note: Do not use any parameters inside this function.

Improving JavaScript performance

Keep the following tips in mind while creating your JavaScript to improve the performance of your code:

- Always use the var statement to access variables.
For example, var message = "HELLO";

- Always define JavaScript arrays to be of specific size, or with a list of initial elements.
  For example, var s_objs = new Array(10);

- Avoid using the `eval` and `new` functions.

- Avoid using the `with` statement.

- Always specify classes with a fully qualified name. Use "Packages" keyword when using such classes. Refer to Invoking Java methods in JavaScript on page 50.

### Debugging the JavaScript code using writeLog

This section describes how to debug your JavaScript code and restart suspended worksteps.

You can use the `writeLog` method to print the value of JavaScript variables, or any other information, to help you debug your JavaScript code.

**Note:** As JavaScript `writeLog()` enables logging messages in debug mode only, do not write messages into bizlogic.log until you set the BP Server log level to debug.

The following is an example of how to use the `writeLog` method:

```javascript
var name = jst.getDataSlotValue("empName");
var salary = jst.getDataSlotValue("empSalary");
jst.writeLog("DS empName+empSalary are read from server: " + name + ";" + salary);
jst.putDataSlot("empName", name);
jst.putDataSlot("empSalary", salary);
jst.writeLog("update DS empName is ok");
jst.writeLog("update DS empSalary is ok");
```

The `writeLog` method attaches the process instance and workstep names as a prefix for each message printed in the log file, as shown below:

```javascript
JSCRIPT:processInstanceName->workstepName:<msg>
```

For example, output in the `bpsserver.log` file would look similar to the following:

```plaintext
JSCRIPT:h2#563->ArrangeInterview: update DS empName is ok.
```

You can use this to identify the messages when you have JavaScript code operating in multiple worksteps and processes at the same time. Remember to remove these statements when you have finished debugging your code.

### Publishing the application

Publishing an application involves reviewing the default directories to make sure the necessary files are properly stored under the `OEBS_HOME` directory, installing the process template, and running the application.
Installing the application

You may install applications in the Administration page in Business Process Portal. For more information about installing applications, see the Business Process Portal Administrator’s Guide.

If your application does not have rules, then you can also install applications in BP Server Admin. The following directions enable you to install applications through BP Server Admin on Windows.

Note: Prior to starting BP Server Admin, you must start the BP Server.

To publish the Assign_A_Task_VX application:

1. Select Start > Programs > Progress > OpenEdge > BP Server > Admin Consoles > Business Process Console.
2. Execute the connect [user name] [password] command.
   Substitute a valid user name and password. This connects to the BP Server.
3. Execute the getprocesslist command.
   This displays the list of all currently installed process templates. If you are running this application for the first time, then you can not see Assign_A_Task_VX in the list of process templates currently installed. If this application was run before, then however, you should see Assign_A_Task_VX in the list.
4. Create the Business Process Server process from the XML file using the following commands.

   create Assign_A_Task_VX.xml
   install Assign_A_Task_VX

   If you invoke getprocesslist again, then you should see the Assign_A_Task_VX process with the status "Installed Process Template."

Running the application

Once the Assign_A_Task_VX application is published and installed, its name appears in the Application List page of the Home module in Business Process Portal. You can run the application by simply clicking its name. Run the application to verify that the application performs as intended. With appropriate user access permissions, you can also view the status of the process and worksteps.


If a class that is already loaded, is modified or is moved to a new location, then the class needs to be loaded again. This is called dynamic class loading, and BP Server supports it in the current release. For more information, refer Business Process Server Developer’s Guide.

Removing and reinstalling an application

If you run the application and then want to debug or change the process template, you must first uninstall the application and then modify the template in Business Process Studio. For more information about uninstalling and reinstalling applications, see the Business Process Portal Administrator’s Guide.
Alternatively, you can remove an existing application process template by entering the following command in BP Server Admin:

```
removePT Assign_A_Task_VX
```

After modifying the process template file, reinstall the application from BP Server Admin using the `install` command.

### Suspending/Resuming installed applications

To install a new version of an application, you must first suspend the previous version of the application before installing the new version. For more information about suspending and resuming applications, see the *Business Process Portal Administrator's Guide*.

Alternatively, you can invoke the `suspend()` method in the Process Template SVO in the BP Server API to suspend an installed process template. If you use the `createVersion` method, then this is done internally.

**Note:** When you suspend a process template using the `suspend()` method in the Process Template SVO, you cannot create new instances on the process template. If instances of an already suspended process template require creation, then use the `resume()` method in the process template SVO for that purpose.

### Resuming the suspended workstep

If the execution of the JavaScript code fails, then the workstep is suspended. The failure could be due to an invalid dataslot name or other JavaScript coding error. Either way, you cannot restart the workstep until these errors are fixed and the JavaScript code is updated and compiled. Use the BP Server Admin command `ResumeWS`.

JavaScript execution can fail after executing the `jst.putDataSlot()` method partially. In this case, the execution has not completed but still has been successful in partially changing the state of the process instance. If you are not confident of the consistency of the dataslot values, then it is better to remove the suspended process-instance and create a new one.
Using an LIST dataslot to invoke multiple subprocesses

This chapter, explains to create, as well as pass and retrieve data from, multiple subprocess instances, using an LIST dataslot as the subprocess performer.

For details, see the following topics:

- Reviewing the publisher
- Running the application in Business Process Portal

Reviewing the publisher

This chapter uses three sample applications, Publisher, FictionAuthor, and NonFictionAuthor, to illustrate the use of the LIST dataslot as the performer of a subprocess workstep. The LIST dataslot can then create parallel instances of multiple subprocesses. The Publisher application is the parent BP Server application, while FictionAuthor and NonFictionAuthor are the subprocess applications.

The Publisher process contains the "Invoke Subprocess" subprocess workstep, performed by a LIST dataslot named "SubProcessNames". This LIST dataslot contains five values (as shown in Figure 7: Publisher application worksteps on page 60), and each value refers to an available subprocess that contains the dataslots required by the parent process. In the example shown in Figure 7: Publisher application worksteps on page 60, OpenEdge BPM Designer understands that five instances must be created —that is, three instances of the FictionAuthor subprocess and two instances of the NonFictionAuthor subprocess.

You can view these process template files in OpenEdge BPM Designer by importing them from the Workspace_Home directory.
Defining the main process worksteps

As shown in the following figure, the Publisher application has three main worksteps:

- **View publisher information** — Displays the values of the first five data slots (described in Table 8: User-Defined data slots in the publisher application on page 61). For an illustration of how this workstep appears in Business Process Portal, see Figure 7: Publisher application work steps on page 60.

- **Invoke subprocesses** — The subprocess performer, the @SubProcessNames LIST data slot, creates three instances of the FictionAuthor application, and two instances of the NonFictionAuthor application. This LIST data slot passes the values of AuthorIDs, AuthorNames, BookTitles, PublisherName, and publisher address data slots by mapping them to data slots in the subprocess instances. For example, the value in the AuthorIDs data slot in the parent process is mapped to the AuthorID data slot in the subprocess instances. If the data slot name in the parent process is identical to the data slot name in the subprocess, then no mapping is required. This workstep also stores the values returned from the subprocess instances, in the EstimatedDays and ExpectedRoyalty data slots.

  **Note:** The data slots in each subprocess that receives data from the parent process must have the same name. In this case, for example, all subprocesses must contain data slots named AuthorID, AuthorName, and BookTitle.

- **Get results** — Displays the value of the EstimatedDays and EstimatedRoyalty LIST data slots, returned by each of the subprocess instances.

Figure 7: Publisher application work steps
Defining the main process dataslots

After importing all three applications in BPM Designer, open the **Dataslots** content pane tab of the Publisher application to review the user-defined dataslots. The Publisher application contains three LIST dataslots — "AuthorIDs", "AuthorNames", and "BookTitles"—which are provided as input dataslots to the subprocess instances.

Each value of the LIST dataslot is provided as input to the subprocess instance, in the order of their listing. For example, the "AuthorIDs" dataslot has values, "G G Marquez" and "A Roy." So, the first subprocess instance receives the value, "G G Marquez" while the second subprocess instance receives the value, "A Roy." The remaining three subprocess instances receive null values.

User-Defined dataslots in the publisher application

The Publisher application’s user-defined dataslots, along with the other dataslots, are described in the following table.

Table 8: User-Defined dataslots in the publisher application

<table>
<thead>
<tr>
<th>Dataslot</th>
<th>Values (in order of listing)</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>AuthorIDs</td>
<td>G G Marquez, A Roy</td>
<td>This LIST dataslot contains two Author IDs values, which are individually passed as a String to each of the first two subprocess instances. The remaining three subprocess instances receive null values.</td>
</tr>
<tr>
<td>AuthorNames</td>
<td>Gabriel Garcia Marquez, Arundhati Roy, Yann Martel, A P J Abdul Kalam</td>
<td>This LIST dataslot contains four author names as values, which are individually passed as a String to each of the first four subprocess instances. The remaining subprocess instance receives a null value.</td>
</tr>
<tr>
<td>BookTitles</td>
<td>One Hundred Years of Solitude, The God Of Small Things, Life of Pi, Ignited Minds, The Best Quizzes of Derek O’Brien</td>
<td>This LIST dataslot contains five values for book titles, out of which five are individually passed as a String to each of the five subprocess instances.</td>
</tr>
<tr>
<td>PublisherName</td>
<td>Penguin Publishers</td>
<td>This is the only CHARACTER dataslot, and its hard-coded value is passed as a String to all five subprocess instances.</td>
</tr>
<tr>
<td>publisheraddress</td>
<td>New Delhi, London, New York, Chicago, Paris</td>
<td>This LIST dataslot contains five values for publisher addresses, which are individually passed as a List value to each of the five subprocess instances.</td>
</tr>
<tr>
<td>SubProcess Names</td>
<td>FictionAuthor, NonFictionAuthor, FictionAuthor, NonFictionAuthor, FictionAuthor</td>
<td>This LIST dataslot is used as the subprocess performer and is used to create three instances of FictionAuthor process and two instances of the NonFictionAuthor process.</td>
</tr>
</tbody>
</table>
Chapter 4: Using an LIST dataslot to invoke multiple subprocesses

<table>
<thead>
<tr>
<th>Dataslot</th>
<th>Values (in order of listing)</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>EstimatedDays</td>
<td>This LIST dataslot displays the corresponding value returned from each of the subprocess instances, in the form of a List box.</td>
<td></td>
</tr>
<tr>
<td>ExpectedRoyalty</td>
<td>This is another LIST dataslot that displays the corresponding value returned from each of the subprocess instances, in the form of a List box.</td>
<td></td>
</tr>
</tbody>
</table>

Defining an LIST dataslot

To define the LIST dataslot to be used as the subprocess performer (in this instance, SubProcessNames):

1. From the Dataslots content pane tab of the Publisher process, click New.
2. In the New Dataslot dialog box, type a name in the Name box and select LIST from the Type drop-down list. Type a value in the Initial Value box and click Add to add the value to the Values listing. Similarly, add the other values.
3. Click OK to define the dataslot.

Assigning an LIST dataslot as the subprocess performer

To assign an LIST dataslot as the subprocess performer:

1. From the Performers content pane tab, click New. Select Sub-Process as the performer type and click OK.
2. In the Performer dialog box, click the ellipsis button beside the Sub-Process box and then select the Use a Dataslot option to display the Use a Dataslot dialog box.
3. Select the dataslot (in this case, SubProcessNames) and then click OK to add the subprocess performer.
4. Select the Indexed checkbox to mark this subprocess as indexed.

**Note:** The Indexed feature ensures that the values in the parent process' LIST dataslot are not overwritten, when it receives data from multiple subprocesses, and that all the data is preserved in the parent process. Also, if the Indexed checkbox is not selected, then each subprocess that is completed and submits data to the parent process overwrites data from subprocesses that were completed earlier. For this reason, we recommend using a non-Indexed subprocess only when no data is to be returned or when overwriting of data does not matter.

5. Click OK.

Adding a Subprocess workstep to the process template diagram

To add the Subprocess workstep to the process template diagram:

1. Click the Assign participants link in the Tasks pane.
2. Expand the "Sub-Processes" type and then "External References" to display the added subprocess.
3. Drag the subprocess icon on the content pane.
Alternatively, you can modify the subprocess performer, using the Properties view for the Subprocess workstep. For detailed information regarding subprocess performers, see the OpenEdge Getting Started: Developing BPM Applications with Developer Studio.

Running the application in Business Process Portal

If you have administrator privileges, then you can publish all the three applications from Progress Developer Studio for OpenEdge and then install them in Business Process Portal. For information about installing applications, see the Business Process Portal Administrator's Guide.
1. Create an instance of the Publisher application. The first workstep, "View Publisher Information being passed to subprocesses", is seen in the following figure.

**Figure 8: View publisher information workstep**

![View publisher information workstep](image)

2. Click **Complete** to complete this task. This invokes the **Invoke Subprocesses** subprocess workstep, which creates five instances of the subprocess applications. The following figure shows the instances in the **My Tasks** page of Business Process Portal.

**Figure 9: Subprocess instances**

![Subprocess instances](image)
You can complete these tasks, in the order of your choice. For example, click the task, "Provide information back to the Publisher" belonging to FictionAuthor (129) instance to display the following page.

**Figure 10: FictionAuthor page**

![FictionAuthor page](image)

As this is the first subprocess instance to be created, it displays the datalot values retrieved from the corresponding List dataslots.

3. Modify the values in the **Estimated days** and **Expected Royalty** boxes from 10 (default for all FictionAuthor instances) to 20.

4. Click **Complete** to complete this task.

5. Next, complete the FictionAuthor (131) instance task and then the remaining tasks.

6. On completion of all the tasks (including the NonFictionAuthor tasks), the **My Tasks** page now displays the final task, "Get results back from subprocess", of the Publisher applications. Click the task name to display the following "Get results back from subprocess" page.

**Figure 11: Get results page**

![Get results page](image)

This page displays the Estimated days and Expected royalty values returned from each of the subprocess instances. As the FictionAuthor (278) instance task was completed first, the values returned from this instance is displayed at the top of the respective list box values.
Using custom dataslots

This chapter describes how to create customized dataslots that can contain such complex objects as a table, a bar graph or any group of related data presented as a single entity.

For details, see the following topics:

• Reserved dataslot names
• Defining custom dataslots with complex objects
• Using form validation

Reserved dataslot names

The following table contains a list of reserved dataslot names. Do not use any of the words in this list as a name for a new dataslot and do not redefine any of the names or use any of these words when redefining a dataslot.

Table 9: Reserved dataslot names

<table>
<thead>
<tr>
<th>BP ServerSID</th>
</tr>
</thead>
<tbody>
<tr>
<td>ptName</td>
</tr>
<tr>
<td>__webAppName</td>
</tr>
<tr>
<td>__uniqueld</td>
</tr>
</tbody>
</table>
Defining custom dataslots with complex objects

Custom dataslots can be used to define the presentation of a complex object that cannot be accomplished with standard dataslot types. For example, it might be required to create a custom dataslot in order to include a table or bar graph in a Web application. The custom dataslot then presents a complex object in an interface page which is seen by the user when the application is launched in the user’s Web browser.

Use any of the following methods to define custom dataslots that can then display complex objects and change the presentation of an interface:

- Default or custom factory (a type of adapter) to create and initialize a custom dataslot (see Using factory-type adapters on page 68).
- The getPresentation() method (see Using the getPresentation function on page 73).
- The Template-based interface generator (see Using template-based interface generator for custom dataslots on page 74).

Using factory-type adapters

You can use a default or custom factory (a type of adapter) to create and initialize a custom dataslot that displays a complex object in a customized interface.

Defining custom dataslots

To define custom dataslots:
1. Open the process template for a Web application in OpenEdge BPM Designer and click the **Dataslots** content pane tab. To create a dataslot, click **New** to open the **New Dataslot** dialog box.

2. Define a new CHARACTER dataslot type to hold the className for the custom dataslot — in this case assign it the name of **MyClass**—and select **CHARACTER** from the **Type** drop-down list. In the **Initial Value** box, type `className` as the custom object to be created to correlate the Factory information with the dataslot.

3. Similarly, define a new OBJECT dataslot type and assign it the name, **MyTable**.

### Initializing the **MyTable** custom Object dataslot

You must initialize the **MyTable** custom Object dataslot before you can use it in the process template.

**To initialize the custom object dataslot:**

1. From the **Assign participants** link in the Tasks pane, drag the **New Adapter** icon to the process template to open the **New Adapter** dialog box.
2. Enter a unique name for the new adapter in the **Name** box, in this case **MyTableAdapter**.
3. In the **Class** box, define the class that invokes the adapter containing the objects (for instance, `com.savvion.BPM.Workflow.beans.DsPrototypeFactory`).

**Figure 12: Defining factory adapter properties**

![Figure 12: Defining factory adapter properties](image)

---

**Note:** The **Method** box is not enabled. You are only able to use the commit method for BPM Workflow adapters.

4. Click **OK** to add the adapter workstep.
5. From the **Properties** view for this workstep, use the **Dataslots** tab to specify the input and output dataslots.
   a) Click **Add** to open the **Select Dataslots** dialog box.
   b) Select the **MyClass**, **MyTable**, and `queryString` dataslots and then click **OK** to add these dataslots to **MyTableAdapter**.
c) For the MyClass and queryString dataslots, click in the Output from Adapter column, and select false in the respective cells.

d) Similarly, specify the MyTable dataslot as output to the new adapter by selecting false in the Input to Adapter column.

The Properties view for the MyTableAdapter workstep now indicates the input and output dataslots you have just defined.

Figure 13: Inputs and outputs in the properties view

6. To map the input custom dataslots:

a) Select Advanced option from the drop-down list.

b) Select the MyClass custom dataslot and click Modify, to open the Field Properties dialog box.

c) Click the ellipsis button next to the Adapter input(s) box, to open the Adapter Input(s) dialog box.

d) Type "className" in the text box provided, and click Add. If necessary, then remove other inputs.

e) Click OK, returning to the Field Properties dialog box, where className is now shown as the Adapter input. You have mapped the MyClass custom dataslot to the className dataslot.

f) Click OK, and return to the MyTableAdapter Properties view, with the Advanced view displayed.

g) Map the queryString dataslot to Query by following Step 6a to Step 6f, substituting queryString for MyClass and Query for className.

Note: When using the DsPrototypeFactory, you must not map output custom dataslots since DsPrototypeFactory handles this mapping.

7. Click File > Save or the Save icon to save the changes.

The dataslot can then be used as input in any of the subsequent activity worksteps in the process template. The activity workstep recognizes the dataslot as a custom slot (through reflection technique) and invokes its getPresentation() method to get the presentation for the dataslot.

Note: For backward compatibility with previously developed applications, object dataslots defined with the full class name of the object or with UserDefinedDS in the Value box are also valid. UserDefinedDS distinguishes the object dataslot with a custom interface. Use an Adapter (MyTableFactory) to initialize the dataslot.
Initializing custom dataslots

For most cases involving custom dataslots, the default DsPrototypeFactory is sufficient to initialize dataslots. In cases where an object requires a complex initialization which cannot be performed by ordinary set methods, you must provide a custom factory to create and initialize the dataslot. We have provided the DsPrototypeFactory class as the default factory class for all custom dataslots so that users are not required to write a factory class for every custom dataslot. The factory can only create one object at a time. Define an input for every property of the customized object to be created.

The following sample code provides an implementation of the MyTableFactory Adapter that can be used to initialize the MyTable custom dataslot.

**Note:** The MyTableFactory class (a subclass of GenericAction) includes the attributes Query and borderThickness, and the commit() method for initializing the MyTable custom dataslot. Any number of attributes can be defined as long as the commit() method is modified to include the code to initialize them.

```java
import com.savvion.BPM Workflow.beans.*;
public class MyTableFactory extends GenericAction {
    public static String QUERY = "Query";
    public static String BORDERTHICKNESS = "borderThickness";
    public MyTableFactory() {
    }

    public int commit() {
        String query = getPropString(QUERY);
        String borderThickness = getPropString(BORDERTHICKNESS);
        MyTable dbtable = new MyTable();
        dbtable.setQuery(query);
        dbtable.setBorderThickness(borderThickness);
        setPropBSObject("MyTable",dbtable);
        return 1;
    }
}
```

Custom dataslot sample code

The following sample code provides the MyTable class that implements a dataslot with custom interface.

**Note**

The MyTable class (a subclass of BSObject) implements the set methods for all the attributes defined for the dataslot. Additionally, this class implements the getPresentation() method for rendering dataslot presentation. If the inout parameter in the getPresentation() method is False, then the dataslot is input only, and it renders a read-only presentation. If the inout parameter in the getPresentation() method is True, then it renders an editable presentation. This method must return a string which represents the presentation for the dataslot.

**Sample code**

```java
package com.savvion.BPM Workflow.beans;
import java.util.*;
import java.net.*;
import java.sql.*;
```
public class MyTable
    extends BSObject
{
    public static final String NL = "\n";

    ResultSet rs;
    ResultSetMetaData rsmd;
    int column;

    private String Query;
    private String DBString;
    private String DBJdbcDriver;
    private String DBUsername;
    private String DBPassword;

    public MyTable()
    {
    }

    public void setDBUsername(String dbUsername)
    {
        DBUsername = dbUsername;
    }

    public void setDBPassword(String dbPassword)
    {
        DBPassword = dbPassword;
    }

    public void setDBJdbcDriver(String dbJdbcDriver)
    {
        DBJdbcDriver = dbJdbcDriver;
    }

    public void setDBString(String dbString)
    {
        DBString = dbString;
    }

    public void setQuery(String query)
    {
        Query = query;
    }

    private String getQuery()
    {
        return Query;
    }

    public String getPresentation(boolean inout, Hashtable pres_args)
    {
        if (inout == false)
        {
            String res = " Query: "+getQuery();

            SQLSession sqlsession = new SQLSession();
            try
            {
                sqlsession.startSQL();
                rs = sqlsession.statement.executeQuery(Query);
                rsmd = rs.getMetaData();
                column = rsmd.getColumnCount();

                res+="<table border=2">"+NL;
                res+="<tr>";
                for(int i=0;i<column;i++)
Using the `getPresentation` function

As an example of dataslots with a custom presentation, consider a case where as part of the presentation of a workstep, the result of a database query needs to be displayed as a table. Without support for dataslots with custom presentation, the only way to accomplish this task would be to customize the generated JSP page to include code to display the result of the query as a table. With the `getPresentation()` function, you can create a dataslot (for example, MyTable) and then use the MyTable dataslot as input to any workstep which is to display the data as a table. The custom dataslot does not support an easy mechanism for entering data; it is intended to be only used for presentation.

Using `getPresentation()` with custom dataslots

You can use the `getPresentation()` method to generate the interface presentation for a custom dataslot. When a custom dataslot is passed as an input, you must only call the `getPresentation()` method. To ensure compatibility with previous code, the custom object base class (BSObject) implements this function as follows:

```java
/**
 * get DS presentation. Present it as input if inout = false
 * or output (editable) if inout = true
 */
```
public String getPresentation(boolean inout, Hashtable presAttrs)
{
    if(inout == false)
        return getHTML();
    else
        return setHTML();
}

Retrieving data from custom dataslots passed as outputs
In the previous section, it was necessary to only call the getPresentation() method of the custom dataslot that is passed as an input.

1. To retrieve data from a custom dataslot passed as output, Business Process Server runtime automatically performs the following:

2. Calls the getPresentation() method to get and display the presentation for that custom dataslot.

3. After submitting the form that contains the custom dataslot, calls the doReqProcessing() method of the custom dataslot.

4. The doReqProcessing() method updates the data that the user entered in the form generated by getPresentation() back into the custom dataslot object, which in turn Business Process Server stores into a data storage mechanism (currently the BPM Workflow bean).

See the following example:

public BSObject doReqProcessing(HttpServletRequest request)
{
    String name = null;
    String addr = null;
    name = request.getParameter("name");
    addr = request.getParameter("address");
    setName(name);
    setAddress(addr);
    return this;
}

Using template-based interface generator for custom dataslots
You can use the BPM CustomUI API API to define the presentation of the custom dataslot as a template. BPM CustomUI API API, which is available to all Web Applications, enables users to customize templates with tags, which are the dynamic part of the template. Users can customize the template as an HTML form.

The implementation of the getPresentation() method can use the setTag() method to describe how the tags in a template should be replaced, and can use the getPage() method to render the presentation for the custom dataslot.

Following is sample implementation of a getPresentation() method that sets the dataFileName tag in the getBarchart template. Note the bsBean is the instance of BPM CustomUI APIBean that is passed as a parameter to the custom dataslot.

public String getPresentation(...)
{
    bsBean.setTemplateDirectory(bsBean.getSBMRootDir() + "/BPM Workflow/common/templates/");
    bsBean.setTag("dataFileName", dataFileName);
    return bsBean.getPage("demo", "getBarchart");
}
Using form validation

We recommend that users perform form validation in the browser on the client side, before data is sent to the server. To ensure that data is entered into a particular field, you can select Required in the New Dataslot dialog box for the dataslot that generates the field in presentation. See OpenEdge Getting Started: Developing BPM Applications with Developer Studio for a description of this procedure. Once designated as Required, an asterisk appears to the right of the label in the presentation.

Business Process Server supports form validation by using JSP tags that automatically generates client-side Java form validation. Users can only see the tags in the generated JSP pages, and the lengthy Java code that existed in earlier BPM Workflow versions are not displayed.
Chapter 5: Using custom dataslots
Developing Business Process Server adapters

Adapters are non-human performers of worksteps that are typically a code that connects Business Process Server to databases, third party softwares, or other systems.

An adapter allows Business Process Server applications to invoke external Java classes residing on the BP Server (on remote/local machines) to perform a task. For example, when you add the E-mail Managed Adapter included with the Business Process Server installation to the process template, Business Process Server automatically inserts the Java class and method as the performer of the Adapter workstep, enabling task automation. In addition to the predefined adapter, you can develop and write your own adapters.

The BP Server provides the following functionalities:

- **Single method to obtain input data slot values.** As an application developer, you can obtain the values of all the input data slots by defining a single method `setAllInputDataslots(Hashtable)`.

- **Single method to send output data slot values.** You can send all the output data slot values to the BP Server in a single method call by implementing `Hashtable getAllOutputDataslots()`.

- **Obtain run-time process information from the adapter.** You can obtain run-time information about a process by implementing `setProcessContextData(Hashtable)`.

- **Dynamic reloading of adapters.** You can dynamically reload the adapter classes from BP Server without having to stop the BP Server. This is not supported when the Adapter workstep has an Object data slot (Minimum JDK Requirement: JDK 1.2.).

For details, see the following topics:

- About BP Server adapters
- Developing adapters
About BP Server adapters

Adapters are objects executed remotely and run on a separate, external JVM. The communication for a remote adapter in BP Server is performed using the JNDI protocol. This protocol is specified during the design time on the performer tag of the Adapter workstep by the following syntax:

```
jndi //machinename/jndiname of ejb
```

where machinename is the machine name where the object resides, and jndiname is the name by which the EJB is binded to the JNDI registry.

**Note:** To use this feature with WebLogic application server, ensure that the WebLogic security settings are configured as described in the "Modifying WebLogic Security Settings" section of the Business Process Server Developer’s Guide.

Adapter worksteps in BPM Designer are of two types: synchronous and asynchronous. By design, all Adapter worksteps in Web applications are synchronous.

**Synchronous adapters**

For synchronous Adapter worksteps, when you activate the adapter, the adapter calls the **set** methods, the **perform** method, and finally the **get** method. No matter how long the adapter execution takes, the BP Server and BPM Workflow engines wait for the task to finish.

The synchronous adapter is easy to create and manage. When finished, the dataslots results are consistent and passing information to other adapters is safe. Ensure that the dataslots (which the adapter is modifying) are modified before anyone else looks at them.

**Asynchronous adapters**

For asynchronous adapters, BP Server calls the adapter but does not wait for the adapter to finish its tasks. Use this feature for an adapter which takes a long time to complete (for example, for benchmarks), since the connection is closed between the workflow engine and the adapter.

**Note:** For further information on creating and using asynchronous adapters, see the OEBPS_HOME\BPServer\examples\AsyncAdapterTest.zip file.

The adapter requires certain information about BP Server, the process template, and the work item name. For this, define the method `PAKcallerID()` in the adapter class.

```java
public void PAKcallerID(String processInstanceName, String workstepName, java.util.Properties bizLogicHost);
```
where `PIName` is the name of the process instance and `WSName` is the name of the workstep. `BPServerLocation` has the following keys:

- **APPSERVER** — The type of application server, such as WebLogic, WebSphere.
- **JNDI_NAME** — The JNDI name of BP Server by which it is binded to the JNDI of the application server.

The following keys are used to get the Initial context:

- `INITIAL_FACTORY`: the initial context factory for the appserver
- `PROVIDER_URL`: the url where the bizlogic server is binded to the jndi
- `CREDENTIALS`: the password to connect to the appserver
- `PRINCIPAL`: the user name to connect to the appserver

To get the BP Server remote object, first obtain the initial context factory and then locate the BP Server home object using the jndi binded name. Invoking `create()` on the home object gives the BP Server remote object.

The `PAKcallerID` method is called before BP Server breaks the connection to the adapter. This information is required to complete a task.

**Note:** For custom asynchronous adapters with output dataslots, the `completeCallerWS()` method can be called from the `execute` or `PAKcallerID()` method. For these adapters, the `getAllOutputDataslots()` or individual getter methods are never invoked.

BP Server calls this method to set the process template name, work item name, and the JNDI information of BP Server. If the adapter has no output dataslots, then the adapter is completed and the application continues to the next workstep.

**Note:** An asynchronous adapter may behave like a synchronous external adapter when it contains output dataslots.

Asynchronous adapters are of two types:

- **With output dataslots** — With output dataslots, an asynchronous adapter behaves like a synchronous adapter; that is, it waits for the asynchronous adapter class to complete the adapter.
- **Without output dataslots** — In this case, the next workstep is activated immediately after activating this adapter workstep.

To complete the asynchronous adapter that has output dataslots, call the function `completeCallerWS()`. The parameters are BP Server session, process template name, work item name, and a hashtable that contains dataslot values for you to update before completing the adapter.

Only the asynchronous adapter calls the BP Server.

**Developing adapters**

An adapter is a piece of Java code that connects Business Process Server to external systems or performs complicated calculations, including databases, third party software tools, or hardware devices. The most important requirement is that external systems must have an API with which the adapter can communicate.
This API can be in C, C++, Java, or other languages/formats supported by Business Process Server. If the API is in Java, then you invoke the Java methods from the Java adapter. Note that it might be required to include the JAR files of the external system in your CLASSPATH environment variable.

**Note:** If the API is in C or C++, then you require special Java wrappers (JNI wrappers) for the Java adapter to invoke C/C++ functions.

---

**Defining set/get methods (setters and getters)**

BPM Workflow and BP Server use the same set/get mapping methods, but use different dataslot naming conventions. The examples in this chapter follow the naming conventions used in BP Server.

**Note:** To see how to define set/get methods in the Adapter workstep, refer to "Mapping Dataslots in an Adapter Workstep" section of the *OpenEdge Getting Started: Developing BPM Applications with Developer Studio*.

BP Server uses the default mapping setX and getX for the methods (unless setAllInputDataslots and getAllOutputDataslots are defined in the adapter) corresponding to a dataslot named "X" used in an adapter. You can, however, use your own mapping. For example, in the Emailler adapter you might choose to associate the methods setEmailSubject, setEmailAddress, and appendEmailBody with the Subject, Recipient, and Content dataslots respectively.

In the Adapter workstep properties, select the dataslots as input or output to the adapter. For each input dataslot, one set method of the form set[dataSlot_name]() should exist in the adapter code (public void setpath [String value], where path is a dataslot of type CHARACTER). Also, for each output dataslot, one get method of the form get[dataSlot_name]() should exist (public String getDocURL(), where DocURL is a dataslot of type CHARACTER).

An example of a DATETIME-TZ and DECIMAL dataslot follows.

**DateTime-TZ dataslot example**

```java
public void setDsDate(java.sql.Timestamp value) {
}
public java.sql.Timestamp getDsDate() {
}
```

where the return type for date is java.sql.Timestamp.

**DECIMAL dataslot example**

```java
public void setDsDecimal(java.math.BigDecimal value) {
}
public java.math.BigDecimal getDsDecimal() {
}
```

where the return type for decimal is java.math.BigDecimal.

Finally, the performer method can be any name with the return type "void." This method executes the logic/work of the adapter.

**Note:** You can use either set/get method for each individual dataslot or implement the setAll/getAll methods. But they cannot contain the individual getter method for input dataslots or the getAllInputDataslots(Hashtable) methods implemented from the same adapter class. For more information about setAll/getAll methods, see Using setAllInputDataslots and getAllOutputDataslots on page 81.

**Example**

The `TAXmeter` adapter calculates the tax for specific amounts of money, which requires the following two inputs:
• Amount of money
• Tax bracket

The adapter produces one output: Result

As the adapter developer, you can either use the getResult method to calculate the return value, or define an intermediate method which performs the calculation and sets the value of an internal variable for return by the getResult method.

Adapter interface

Business Process Server provides an interface called AdapterInterface under common.jar. The adapter can extend this interface to extend the new functionality. The interface is provided by the following segment code:

```java
import java.util.*;
/**
 * interface with methods that need to be implemented in Adapter workstep
 * to take advantage of the new features supporting EP_LIBRARY.
 */
public interface AdapterInterface
{
    /*
    * updates all the output dataslots passed in a hashtable
    */
    public Hashtable getAllOutputDataslots();

    /*
    * Returns all the input dataslots in a hashtable
    */
    public void setAllInputDataslots (Hashtable inputDS);

    /*
    * Returns the information related to the current workstep in a
    * hashtable.
    */
    public void setProcessContextData (Hashtable processCtx);
}
```

Note: Include the common.jar file from OEBPS_HOME\lib in the CLASSPATH.

Using setAllInputDataslots and getAllOutputDataslots

In addition to using the set/get method to develop adapters, the BP Server provides a way to invoke a single method to either retrieve or update multiple data slot values at once. The setAllInputDataslots and getAllOutputDataslots methods package all input or output dataslots in one method. These methods send or receive all dataslot values in a hashtable. You may use this feature in place of calling a get/set method for a single data slot value to improve application performance.

Note: You can use either the set/get method for each individual data slot or implement the setAll/getAll methods. But you cannot use the individual getter method for input data slots with the getAllInputDataslots(Hashtable) methods implemented from the same adapter class.

In order to implement and use this feature, the adapter classes must implement the following two functions:

1. setAllInputDataslots (Hashtable ht) retrieves all the input dataslots
This method gets all the input dataslots in the form of a hashtable from the BP Server. The hashtable passed to setAllInputDataslots from the BP Server contains the following information for each input dataslot:

- **Key**: <InputDataSlotMappedName>
- **Value**: <InputDataSlot -Value>

2. `getAllOutputDataslots()` updates all the output dataslots

The adapter classes use this method to pass all the output dataslots along with the new values in the form of a hashtable to the BP Server.

The hashtable passed from `getAllOutputDataslots` to the BP Server inside the adapter contains the following information for each output dataslot:

- **Key**: <OutputDataSlotMappedName>
- **Value**: <OutputDataSlot -Value>

If all the output dataslots exist in the hashtable passed from the adapter, then the BP Server updates the output dataslot values with the new values.

**Note**: In both hashtables described above, the hashtable value is passed as a Java object. You can use the Java API to retrieve the data types for the dataslots returned as Java objects. For example, assume that OutputDS1Obj is the Java class passed in value for the OutputDS1 key. You can retrieve the Java data type associated by invoking `OutputDS1Obj.getClass().getName()`. This returns Long, Double, String, or other objects for object dataslots, depending on the Java data type for `OutputDS1`. The `setProcessContextData` method also retrieves this.

### Sample Code

A sample code segment with the previously mentioned methods in the adapter class is shown here:

```java
public void setAllInputDataslots(Hashtable inputDS) {
    //suppose:
    // - inputSlot1 and inputSlot2 are string dataslots.
    // - inputSlot1 is not mapped.
    // - inputSlot2 is mapped to XXX.
    //You can get the two dataslot values using the following code:
    inputSlot1Value = (String)inputDS.get("setinputSlot1");
    inputSlot2Value = (String)inputDS.get("XXX");
    ...
}

public Hashtable getAllOutputDataslots () {
    //suppose:
    // -outputSlot1 and XXX are string variables in EP
    // and they are both the output of the EP.
    // you can put the two output values into return
    // hashmap using the following code:
    Hashtable outputDS = new Hashtable ():
    outputDS.put("getoutputSlot1", outputSlot1);
    outputDS.put("XXX", XXX);
    ...
}
```

### Notes

Please note the following:

- `Hashtable inputDS` represents the hashtable with variable name "inputDS" that contains all the input dataslots.
• inputSlot1 and inputSlot2 represent application dataslots.

• getAllOutputDataslots throws an exception for missing output dataslots and suspends the adapter.

• You can use the setAllInputDataslots() and getAllOutputDataslots() methods to pass all dataslot values at once between the BP Server and the adapter instead of the individual getter/setter. However, it also supports getter/setter for each dataslot. If setAllInputDataslots() or getAllOutputDataslots() is defined in the adapter, then it is used. Otherwise the BP Server tries to use the dataslot getter/setter.

• If dataslots, which are not declared as output for an adapter, are passed in the hashtable, then the BP Server ignores these dataslots while updating.

• In cases where dataslots values are mapped to new names, both the hashtables described above have mapped names as their keys.

Handling adapter exceptions

If you want to catch exceptions or display error messages thrown from an adapter during its execution, then the methods in the class invoked must throw exceptions in order for the BP Server to display these messages in the case of errors.

When the adapter execution fails and throws an exception to the caller, the BP Server captures that remote exception, logs the message and stack trace, suspends the Adapter workstep, and notifies the Business Process Server administrator by e-mail. Using the stack trace, the user can find exactly which method throws the exception and possibly the line number.

Note: If you do not want an exception thrown, then use a dataslot to assign the status of the application use (depending on the type of application and use).

An Adapter workstep exception is captured by BP Server, and BP Server tries to re-invoke the adapter. After specified re-tries, BP Server generates two events, EP_AFTERBREAK and W_SUSPENDED. Then, according to the requirement, applications can use any of the events to take further action. These events are also captured by BPM Events (or a simple event listener) to trigger corrective actions.

For the getAllOutputDataslots() method, the BP Server throws an exception in the following cases:

• Any of the output dataslots for this Adapter workstep are missing from the hashtable.

• Any data type mismatch in the values are passed from the hashtable.

Creating adapters that call a C function through JNI

To create an adapter that calls a C function using Java Native Interface (JNI), perform the following steps:

1. Write the Java adapter that declares the native methods.
2. Compile the Java class.
3. Generate the header file.
4. Write the implementation of the native methods in C.
5. Compile the header and implementation files into a shared library.
6. Place the shared library in your system32 directory.
Data mapping

Before coding a custom BP Server adapter, review the dataslots mapping to Java data types. The following data types are defined in Business Process Server: CHARACTER, INTEGER (containing Double, and Decimal), INT64, LOGICAL, DATETIME-TZ, LIST, Map, Object, and Business Object. The LOGICAL, INT64, INTEGER, and CHARACTER have one-to-one mapping to Java data types, as listed in the following table:

<table>
<thead>
<tr>
<th>Business Process Server dataslot type</th>
<th>Java data type</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOGICAL</td>
<td>com.progress.lang.Logical</td>
<td>None</td>
</tr>
<tr>
<td>DATE, DATETIME, DATETIME-TZ</td>
<td>com.progress.lang.DateTimeTZ</td>
<td>None</td>
</tr>
<tr>
<td>Document</td>
<td>java.lang.String</td>
<td>The Document dataslot, however, is actually a URL pointing to a document which resides on the Web server. Do not keep a document on the workflow engine, especially when it is large (as it becomes inefficient). However, only this URL is necessary to retrieve the actual document from the Web server.</td>
</tr>
<tr>
<td>LIST</td>
<td>java.util.Vector</td>
<td>None</td>
</tr>
<tr>
<td>Map</td>
<td>java.util.LinkedHashMap</td>
<td>None</td>
</tr>
<tr>
<td>DECIMAL</td>
<td>com.progress.lang.Decimal</td>
<td>None</td>
</tr>
<tr>
<td>INTEGER</td>
<td>com.progress.lang.Integer</td>
<td>None</td>
</tr>
<tr>
<td>INT64</td>
<td>com.progress.lang.Int64</td>
<td>None</td>
</tr>
<tr>
<td>Object</td>
<td>java.lang.Object</td>
<td>Your set() method must assign the object to the proper java class type. BP Server assigns the object, retrieved in the get() method to java.lang.Object. An example of how to use the Object dataslot is provided in the following section.</td>
</tr>
<tr>
<td>CHARACTER</td>
<td>com.progress.lang.Character</td>
<td>None</td>
</tr>
</tbody>
</table>

Example of mapping an object dataslot

Assume you have an Object dataslot called User. To initialize this dataslot, refer a class name to it. You can call it User as well. You can initialize an object dataslot when defining it in BPM Designer—enter "User" as the initial value. Now you need a Java class called User. To pass the dataslot to an adapter, you provide the set method setUser(Object o). Refer to the following sample code:

```java
public class User{
    private User myuser = null;
    public User(){
```
class User has the property myuser which is of type User. The set method, setUser, sends a Java object. This is the method where the object is cast to User and then the perform by method, dowork, uses the properties and the methods of the class User.

The get method returns a Java object; however, this is not mandatory because the BP Server engine casts whatever it retrieves from the adapter to an object.

**Note:** BP Server automatically creates skeleton Java classes with the correct get/set methods.

### Mapping existing methods from a predefined adapter

Business Process Server enables you to map dataslots from an existing application to an adapter’s get/set methods. Use this approach if you have developed an application and then find an existing adapter in the Adapter Library that is exactly what you want, but the get and set methods do not follow the naming scheme that you have selected for your dataslot names. BPM Designer has provided the Advanced setting in the Fields tab in the Adapter workstep’s Properties view to solve this problem. Select the dataslots to map to the get/set methods. For more information, see the OpenEdge Getting Started: Developing BPM Applications with Developer Studio.

### Using the setProcessContextData method

You may implement the setProcessContextData method to obtain information on the current adapter or the currently executed process. The process context information is returned in a hashtable. To utilize this new feature, the adapter class must implement the setProcessContextData(Hashtable ht) function.

The information presented in the following table is available to the adapter class from the hashtable:

**Table 11: Information available to the adapter class from the hashtable**

<table>
<thead>
<tr>
<th>Type</th>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHARACTER</td>
<td>ProcessTemplateName</td>
<td>&lt;ProcessTemplateName&gt;</td>
</tr>
<tr>
<td>CHARACTER</td>
<td>ProcessInstanceName</td>
<td>&lt;ProcessInstanceName&gt;</td>
</tr>
<tr>
<td>CHARACTER</td>
<td>WorkstepName</td>
<td>&lt;WorkstepName&gt;</td>
</tr>
<tr>
<td>INT64</td>
<td>ProcessInstanceID</td>
<td>&lt;ProcessInstanceID&gt;</td>
</tr>
<tr>
<td>INT64</td>
<td>WorkstepID</td>
<td>&lt;WorkstepID&gt;</td>
</tr>
</tbody>
</table>
Hashtable¹ and Hashtable² have "mappedDataslotName" and "originalDataslotName" as key–value pairs. If, for example, you have an adapter with the input dataslots "name" and "address", and output dataslots "salary" and "tax", then the input dataslots are mapped to epName and epAddr respectively. Output dataslots remain the same, and they are not mapped.

To obtain the mapped dataslot names for this adapter class, you must implement the method `public void setProcessContextData(Hashtable processCtx)`. From inside this method, mapped dataslot names for the adapter are obtained as a hashtable.

The line/code `Hashtable inpSlots = processCtx.get("MappedInputDataSlots")` returns the "MappedInputDataSlots" in a hashtable. The returned hashtable inpSlots contains epName, name and epAddr, address as the key–value pairs. These pairs correspond to the mappedName -> epAddr, originalSlotName->address and for the input dataslots of the adapter.

Similarly, hashtable `outSlots = processCtx.get("MappedOutputDataslots")` have salary, salary and tax, tax as the key–value pairs. Since these values are not mapped, the mapped name and the original name remain the same.

### Dynamic reloading of adapters

Business Process Server applications can reload the latest version of the adapter class dynamically from the BP Server without stopping the server. The BP Server is configured to dynamically reload adapter classes from the OEBPS_HOME\ebmsapps directory defined in the oebs.conf file as the parameter oebps.application.home. Therefore, store the adapter classes under the OEBPS_HOME\ebmsapps directory.

The BP Server also dynamically loads JAR files from the OEBPS_HOME\ebmsapps\<process_name>\lib and OEBPS_HOME\ebmsapps\common\lib folders and class files from the OEBPS_HOME\ebmsapps\common\classes, OEBPS_HOME\ebmsapps, and OEBPS_HOME\ebmsapps\common\ejbadapters folders.

### Sample adapter code

The following sections explain how you can connect a Business Process Server process to a database and to a printer.
Connecting Business Process Server to a database

The following is a sample adapter, written in Java, that connects a Business Process Server process to a database using JDBC.

```java
import java.sql.*;
import sun.jdbc.odbc.*;
public class SaveInfo {
    Connection dbConnect = null;
    public SaveInfo() {
        //initialize db connection in the constructor
        new JdbcOdbcDriver();
        try {
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            //connect to db
            dbConnect = DriverManager.getConnection("jdbc:odbc:SQLServer", "sa", "");
        } catch (Exception e) {};
        try {
            Statement stmt = dbConnect.createStatement();
            stmt.execute("/*some sql statement here*/");
        } catch (Exception e) {};
    }
    public void writeToDB() {
        try {
            Statement stmt = dbConnect.createStatement();
            stmt.execute("/*some sql statement here*/");
        } catch (Exception e) {};
    }
}
```

Connecting Business Process Server to a printer

The following is a sample adapter, written in Java, that connects a Business Process Server process to a printer. The example demonstrates classic adapter methods, as well as native methods.

```java
public class PrintFile{
    String m_strFilename;
    String m_strPrinterFullPath;
    long m_lResult =0;
    // classic Adapter methods
    public void setFilename(String s) { m_strFilename = s; }
    public void setPrinterFullPath(String s) { m_strPrinterFullPath=s; }
    public void printFile() {
        if (m_strPrinterFullPath.length() == 0)
            m_lResult = printFile(m_strFilename);
        else
            m_lResult = printFile(m_strFilename, m_strPrinterFullPath);
    }
    public Long getPrinterResult() { return new Long(m_lResult); }
    // native methods
    public native int printFile(java.lang.String strFilename, java.lang.String strPrinterFullPath);
    public native int printFile(java.lang.String strFilename);
    // loading the dll
    static {
        System.loadLibrary("CPrinter");
        System.out.println("CPrinter.dll has been loaded");
    }
}
```

The sample Printer Connection adapter uses four classic Java methods:

- setFileName()
- setPrinterFullPath()
• printFile()
• getPrinterResult()

The setFileName() and setPrinterFullPath() pass the values of the dataslots "FileName" (the name of the file to print) and "PrinterFullPathName" (the full path of the printer) to the adapter. The void method printFile() actually prints the file. The final method passes the values from the adapter back to the process.

The example also uses two native methods for the adapter. The first method specifies the full path name of the printer. The second method works in conjunction with the default printer. Once the shared library is loaded, you can find the implementation of these native methods in the DLL (Dynamic Link Library).

Generating the sample adapter

To generate the sample adapter, run the following commands:

1. javac Printfile.java
   This compiles the Java class.
2. javah -jni PrintFile
   This generates the header file, PrintFile.h, displayed below.

```
JNIEXPORT jint JNICALL Java_PrintFile_printFile__Ljava_lang_String_2Ljava_lang_String_2
   (JNIEnv *, jobject, jstring, jstring);
```

3. Implement the native methods. An outline is shown below.

```
JNICALL jint JNICALL Java_PrintFile_printFile__Ljava_lang_String_2
   (JNIEnv *env, jobject obj, jstring strFilename)
{
  jboolean isCopy;
  const char *filename = (*env)->GetStringUTFChars(env, strFilename, &isCopy);
  ...
  C code to actually print the file
  ...
  if (isCopy == JNI_TRUE)
     (*env)->ReleaseStringUTFChars(env, strFilename, filename);
  return iSuccess;
}
```
Developing BPM Workflow adapters

As mentioned in Developing Business Process Server adapters on page 77, an adapter (also called an external performer or EP) is a Java class that provides a mechanism to integrate third party classes and actions; for example, automating certain functions and tasks performed by a server or other external systems.

Adapters are the “performers” of Adapter worksteps. Depending on the application requirements, you may use a predefined internal system Adapter or develop your own Adapter in Web application development.

For details, see the following topics:

• Using system adapters
• Developing custom adapters
• Compiling adapters
• Deploying adapters
• Debugging adapters

Using system adapters

Business Process Server includes a set of predefined system Adapters that include “general purpose” adapters as well as adapters with more specific functions that allow Web Applications to perform certain BP Server-specific operations.

General purpose adapters used to interact with BP Server include:

• PAKSetDS — Transfers dataslot values from Web Applications back to their BP Server counterparts. This system adapter works internally, and you must not add specific properties to it.
• **PAKGetDS** — Updates the values of dataslots in Web Applications with values received from their BP Server counterparts. This system adapter works internally, and you do not need to add specific properties to it.

• **PAKCreatePI** — Creates a Business Process Server process instance. Use this system adapter at the end of a Web application to trigger a BPM project. For more information, see Using the PAKCreatePI adapter on page 91.

• **PAKUpdateDS** — Saves the state of all BP Server mapped dataslots in the current Web Application. Use this system adapter in a long application, to save information entered up to the location where you insert this adapter. This system adapter works internally, and it is not required to add specific properties to it. For more information, see Using the PAKUpdateDS adapter on page 93.

• **PAKReassignWI** — Enables users to reassign tasks to another performer while still remaining in the process — you must not leave the process and go to Business Process Portal to reassign a work item. For more information, see Using the PAKReassignWI adapter on page 96

### Properties required for system adapters

Each predefined system adapter that has a specific function requires specific input/output properties, and these required properties are listed for each adapter in the following tables.

**Note**

In an Adapter workstep, input properties refer to the information passed to the Adapter. Output properties represent the information you want back from the Adapter.

#### PAKCreatePI adapter properties

<table>
<thead>
<tr>
<th>Dataslot</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ptName</td>
<td>CHARACTER</td>
<td>Name of Business Process Server process template where you want to create the instance.</td>
</tr>
<tr>
<td>bizsite_user/bizsite_password</td>
<td>CHARACTER</td>
<td>BP Server user details (user name and password) used for logging in and starting the session.</td>
</tr>
<tr>
<td>bizsite_priority</td>
<td>CHARACTER</td>
<td>Instance-level priority</td>
</tr>
<tr>
<td>bizsite_instanceName</td>
<td>CHARACTER</td>
<td>Name of the process instance</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Description</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>error</td>
<td>CHARACTER</td>
</tr>
<tr>
<td>Description of any errors</td>
<td></td>
</tr>
</tbody>
</table>

#### PAKReassignWI adapter properties

<table>
<thead>
<tr>
<th>Dataslot</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>performer</td>
<td>CHARACTER</td>
<td>Name of performer to whom the task is assigned.</td>
</tr>
</tbody>
</table>
Using the PAKCreatePI adapter

You can now define a Flow (private) presentation for the Start workstep of a BPM Project. When you right-click the Start workstep and select Open, BPM Designer automatically launches a Web application, with a workflow consisting of a Start workstep, followed by a CreatePI adapter workstep (to be performed by the PAKCreatePI adapter), and an End workstep.

**Note:** PAKCreatePI cannot be used in a Web application that is an interface to an Activity workstep in a BPM Project. It can only be used in a standalone Web application or a Web application that is used as an interface for a Start workstep in a BP Server application.

To use the PAKCreatePI adapter:

1. Click the Create PI workstep (left image, Figure 14: Defining the PAKCreatePI adapter in an adapter workstep on page 91) at the end of the BS_SVBSurvey Web Process (that is, after all significant worksteps have been completed) to display the workstep’s Properties view (right image, Figure 14: Defining the PAKCreatePI adapter in an adapter workstep on page 91).

![Figure 14: Defining the PAKCreatePI adapter in an adapter workstep](image)

2. The Performer box of this workstep displays the PAKCreatePI adapter. If this adapter is not displayed:
   a) Click the ellipsis button beside the Performer box, launching the Performer dialog box.
   b) Click New, to open the Performer dialog box for the Adapter workstep.
c) Enter a name for the Adapter performer in the Name box. Enter the Java class name for the adapter in the Class box. For the PAKCreatePI Adapter, enter `com.savvion.BPM Workflow.beans.PAKCreatePI`.

Figure 15: Performer dialog box for the PAKCreatePI adapter

![Figure 15: Performer dialog box for the PAKCreatePI adapter](image)

**Note:** The Method box is not enabled. You are only able to use the commit method for BPM Workflow adapters.

d) Click OK, returning to the Properties dialog box. The PAKCreatePI adapter class name is now displayed in the Performer box of the Properties dialog box.

e) Click OK, returning to the process template diagram.

Properties of the PAKCreatePI Adapter used to initialize the process (ptName) and report on errors (error) are presented in [PAKCreatePI adapter properties](#) on page 90.

If dataslot names are identical in the Web application and the BPM project, then the mapping of dataslots between these applications occurs by default. This is the case for most of the dataslots in the SVBSurvey sample application. Three dataslots, however, are not identical, and these are: PT_NAME, BIZSITE_USER and BIZSITE_PASSWORD.

**Important:** When using PAKCreatePI Adapter, do not define a user-defined dataslot called ptName, as this is a reserved word that has already been used. Instead, if the name of the WebApp is different than the name of the BP Server application, then you must define another username dataslot of type CHARACTER and map it to ptName when defining the input of PAKCreatePI.
For the PAKCreatePI Adapter to work correctly, the Web application must have dataslots that are mapped to the ptName, bizsite_user and bizsite_password dataslots in the BPM project. To map dataslot names that are not identical, navigate to the Advanced view of the Dataslots tab of the workstep’s Properties view to correctly map the dataslots before using them in the Web/BPServer interaction. Map the dataslots named ptName, bizsite_user, and bizsite_password to PT_NAME, BIZSITE_USER, and BIZSITE_PASSWORD respectively. As a further example, if you have a BPM Workflow CHARACTER dataslot type called myURL that is used to initialize a CHARACTER dataslot type in BP Server called ds1, then you must map myURL to ds1. For more information on mapping dataslots, see OpenEdge Getting Started: Developing BPM Applications with Developer Studio.

When this application is run, the PAKCreatePI Adapter creates a process instance for the user running the application.

3. You can also use the PAKCreatePI Adapter to create a process instance for another user, with the instance appearing in the user’s Task List page in Business Process Portal, by adding two new dataslots to the workstep, as shown in the following steps:

   a) Open the Dataslots tab of the Create PI workstep’s Properties view. The "BIZSITE_USER" and "BIZSITE_PASSWORD" dataslots appear in the table on this tab. Make sure these dataslots are shown as Inputs to the adapter.

   b) If these dataslots are not present, click Add and then New to open the New Dataslot dialog box.

   c) Type BIZSITE_USER in the Name box, and type the user’s name in the Initial Value box.

   d) Click OK.

   e) Similarly, create the "BIZSITE_PASSWORD" dataslot. In this case, type the user’s password in the Initial Value box. Now, when the application is run, a process instance is created for the user identified in the "BIZSITE_USER" dataslot and the instance appears in this user’s Task List page.

Using the PAKUpdateDS adapter

Use the PAKUpdateDS Adapter to save user-entered dataslot information from an incomplete Web application. The PAKUpdateDS Adapter is always related to a Business Process Portal and BPM Designer interaction.

During application development, you can use the PAKUpdateDS system adapter to save the process state during run time without completing the workstep. Use the PAKUpdateDS Adapter when you have an application that is long-lived and you want to save information added up to the location where the PAKUpdateDS is placed (for instance, a specific workstep) in the application process.

An online survey is an example of a long Web application during which it may be necessary to save the process instance state during run-time. There may be many options and choices, and users may not be able to complete the entire process at one time. You can insert the PAKUpdateDS Adapter to save the information entered by the user up to that point. Although the PAKUpdateDS Adapter saves the data you have entered, it does not save the flow and you must proceed through the saved options and choices to arrive at the point where the PAKUpdateDS Adapter was inserted.

To use the PAKUpdateDS Adapter to save information entered into dataslots in an incomplete Web application:
1. Add an Adapter workstep in the process template as an alternative step at the point where you want to save all preceding information.

**Figure 16: Defining the PAKUpdateDS Adapter Workstep**

2. Define the adapter properties (right image, Figure 16: Defining the PAKUpdateDS Adapter Workstep on page 94).

   a) Type a name in the **Name** box.

   b) Enter the Java class name for the adapter. For the PAKUpdateDS, enter `com.savvion.BPMWorkflow.beans.PAKUpdateDS` in the **Class** box.

   **Note:** The **Method** box is not enabled. You are only able to use the `commit` method for BPM Workflow adapters.

   c) Make sure the **Generate Java code** checkbox is not selected. System adapters should not generate a Java file when the adapter workstep is run.

   d) Click **OK** to add the adapter workstep. Open the **Properties** view for this workstep, where the adapter you have added is shown as the performer of the workstep.
3. As shown in Figure 17: Defining the connectors on page 95, make sure that the Activity workstep—in this example, "Question 6"—that represents the point where data is to be saved, has two connectors from it: one to the main flow of the process; the other to the PAKUpdateDS adapter.

**Figure 17: Defining the connectors**

4. Double-click the connector to the PAKUpdateDS adapter workstep, to open its Link Properties dialog box (upper-right image, Figure 17: Defining the connectors on page 95). Select "Submit button" option from the Type list, and type a name in the Label box—in this case, "Save."

5. Double-click the connector to the main flow of the process, to open its Link Properties dialog box (lower-right image, Figure 17: Defining the connectors on page 95). Define the connector as a Submit button connector type and type a label name—in this case, "Continue."

6. You have now added two buttons to the interface page:
   - A Save button that connects to the PAKUpdateDS adapter. In the example, where the adapter output is to an End workstep, you can define the adapter as a Save and Exit mechanism, saving data entered up to that point and exiting to BP Server. Alternatively, you can set the adapter output back to the process, in which case it acts as a Save and Continue mechanism, which means that you have saved the data entered so far, and remain at that location in the process.
   - A Continue button that proceeds with the workflow, without saving any data.

7. Click the Dataslots tab in the adapter’s Properties view. Click Add and then select the dataslots that you require from the Select Dataslots dialog box. Make sure that dataslots selected for this system adapter are input dataslots. The connecting of dataslots to required system adapter properties is performed internally.

The information entered in the selected dataslots is saved up to the point where you inserted the PAKUpdateDS adapter. This information is saved, even though the Web Application has not been completed.
As explained in the above example, the PAKUpdateDS adapter workstep has now been added to the process diagram. Business Process Server now displays buttons labeled "Save" and "Continue" on the "Question 6" workstep interface. If the user clicks **Save**, then the PAKUpdateDS Adapter saves the state of all mapped dataslots up to that point. If the user clicks **Continue**, then the user simply continues in the process.

**Using the PAKReassignWI adapter**

You can use the PAKReassignWI Adapter to reassign tasks to another user during run time without needing to leave the application and go to Business Process Portal and reassign a work item. To do so, follow the procedures that have been described for the PAKUpdateDS Adapter in the previous section (see **Using the PAKUpdateDS adapter** on page 93), but with the following exceptions:

1. To create the PAKReassignWI Adapter performer, enter the following Java class name for this adapter: `com.savvion.BPM Workflow.beans.PAKReassignWI` in the Class box.

2. Instead of labelling the button "Save" as described in Step 4 in the previous section, label it "Reassign".

The PAKReassignWI Adapter reassigns the task to the value of the performer dataslot (listed in **PAKReassignWI adapter properties** on page 90).

**Developing custom adapters**

Use one of the following approaches to develop custom Adapters:

- Define any function to perform the desired activity and extend the GenericAction class (also known as a "BPM Workflow-style" Adapter). For more information, see **Sample adapter code for BPM Workflow-style adapters** on page 97.
Develop BP Server Adapters by implementing the `setAllInputDataslots` and `getAllOutputDataslots` methods to implement a specific interface. For more information, see Sample adapter code for BP Server-style adapters on page 100.

Use Business Process Server’s Adapter and Configuration Mapping Framework to create a custom managed adapter. For more information, see the Managed Adapters Guide.

All Adapter styles are now supported in BPM Designer. You can implement Web Application (formerly BPM Workflow) support of BP Server-style Adapters only for BP Server Adapter version 11.6 and up.

To develop and create a BPM Workflow-style Adapter, extend the `GenericAction` class. The `GenericAction` class has an abstract method, `commit()`, that must be implemented. The `commit` method contains the logic that tells the Adapter what you want it to do. Web Applications in BPM Designer only support the `commit` method. The input data to the Adapter is specified during process design time and is automatically retrieved by the Adapter.

Also, invoke the `getProp<datatype>(dsName)` methods of the `GenericAction` to access the property values. The `setProp<datatype>(dsName, dsvalue)` methods of the `GenericAction` can be used to set property values. The supported datatypes are `String`, `Long` and `Double` of the Number type, Boolean, and Object, as well as Vector and BSOBject.

Place BPM Workflow Adapters (Java class files) in the following directories.

- If the Web application has been published, then save the BPM Workflow Adapter class files in one of the following directories, where AppServer refers to the application server that you are using:
  - `App_Server\...sbm\BPM Workflow\common\classes`
  - `App_Server\...sbm\BPM Workflow\<Application_Name>\WEB-INF\classes`

- If the Web application has not been published already, then save the class files under the following:
  - `OEBPS_HOME\BPM Workflow\<Application_Name>\WEB-INF\classes`
  
  It is also correct to save the files to the following directory:
  - `App_Server\...sbm\WEB-INF\classes`

Sample adapter code for BPM Workflow-style adapters

Use the following sample Adapter code as a template to create custom BPM Workflow-style Adapters.

```java
package com.savvion.BPM Workflow.beans;
import java.util.Enumeration;
import java.util.Vector;
import com.savvion.BPM Workflow.Server.PublicResources;
import com.savvion.BPM Workflow.Server.BPM WorkflowLog;
import com.tdiinc.userManager.UserManager;
import com.tdiinc.userManager.User;
import com.savvion.common.SBM;
import com.savvion.sbm.bizsolo.util.SBMConf;
public class PAKGetDS
extends GenericAction
{
    public PAKGetDS()
    {
    }
    public String[] getInputList()
    {
        Vector names = getAllNames();
        Enumeration e=names.elements();
        String[] snames = new String[names.size()];
        int i=0;
```
while (e.hasMoreElements()) {
    snames[i] = (String) e.nextElement();
    i++;
} return (snames);
}

public int commit() throws Exception {
    String workitemName = getPropString(PublicResources.BL_WORKITEMNAME);
    String ptName = getPropString(PublicResources.BL_PTNAME);
    String mode = getPropString(PublicResources.MODE);
    long workitemId = getPropLong(PublicResources.BL_WORKITEMID);
    String httpSessionId = getPropString(PublicResources.HTTP_SESSION_ID);
    if ("true".equals(getPropString(PublicResources.IS_BIZSOLO_SUBPROCESS)))
        return 0;
    if (workitemName == null && ptName != null &&
        "SLAVE".equalsIgnoreCase(mode)) {
        workitemId = -1;
    }
    String blsid = getPropString(PublicResources.BL_SID);
    if (blsid == null) {
        String msg = "No BP Server connection information received. Aborting ...");
        Log.log(msg, Log.ERROR);
        setError(msg);
        return -1;
    }
    try {
        if (Log.isDebugEnabled())
            Log.log("GetDsPresentationTag :Getting PAKBPM WorkflowBean object using
            values:httpSessionId =" + httpSessionId + ", bLSID=" + blsid + ", workitem id=" + workitemId,
            Log.DEBUG);
            PAKBPM WorkflowBean pakbean = PAKBPM WorkflowBean.getInstance(httpSessionId, blsid,
            workitemId);
            if (workitemId == -1) {
                pakbean.setStartWorkstepDS(ptName);
            }
            String wiperf = pakbean.getWorkitemPerformer();
            boolean isEmailPerf = isEmailPerformer(wiperf);
            if (wiperf != null && isEmailPerf &&
                (PAKBPM WorkflowBean.WI_ASSIGNED ==
                pakbean.getWorkitemStatus())) {
                String sessUser = pakbean.getSessionUser();
                boolean isEmailValid = false;
                String userEmailId = this.getEmailAddress(sessUser);
                if (!wiperf.equals(userEmailId) && !wiperf.equals(sessUser)) {
                    String error = wiperf + " needs to login with the right credentials ");
                    setPropString("error", error);
                    BPM WorkflowLog.log("BPM Workflow_ERR_091", new Exception(error));
                    throw new BPM WorkflowRuntimeException(error);
                }
            } else if (((pakbean.getWorkitemStatus() == PAKBPM WorkflowBean.WI_ASSIGNED) &&
                (pakbean.getSessionUser() != null) &&
                !pakbean.isDelegatePerformer())
                || (pakbean.getWorkitemStatus() == PAKBPM WorkflowBean.WI_COMPLETED)) {
                String error = (pakbean.getWorkitemStatus() ==
                PAKBPM WorkflowBean.WI_COMPLETED) ?
                "PAKBPM WorkflowBean.WI_COMPLETED) : (workitemName + " task has been already
                completed.");
                setPropString("error", error);
                BPM WorkflowLog.log("BPM Workflow_ERR_091", new Exception(error));
                throw new BPM WorkflowRuntimeException(error);
            }
    }
    String[] dsi;
    String[] dso;
    dsi = pakbean.getInputDataslotList();
    dso = pakbean.getOutputDataslotList();
}
if (workitemId > 0) {
    setPropString(PublicResources.BL_PINAME, pakbean.getPIname());
    setPropString(PublicResources.BL_PTNAME, pakbean.getPTname());
    setPropString(PublicResources.BL_TASK_DUEDATE, pakbean.getDueDate());
} else if (workitemId == -1) {
    setPropString(PublicResources.BL_PTNAME, ptName);
}
setPropString(PublicResources.BL_TASK_PRIORITY, pakbean.getPriority());
setPropString(PublicResources.BL_TASK_STARTDATE, pakbean.getStartDate());
setPropString(PublicResources.BL_TASK_INSTRUCTION, pakbean.getInstruction());
for (int i = 0; i < dsi.length; i++) {
    if (Log.isDebugEnabled())
        Log.log("PAKGetDS Input: " + dsi[i], Log.DEBUG);
    try{
        setPropObject(dsi[i], pakbean.getInputDataslotValue(dsi[i]));
    } catch (Exception e) {
        Log.log("Error retrieving input ds name: " + dsi[i] + " error: " + e.toString(), Log.ERROR);
        throw e;
    }
}
for (int i = 0; i < dso.length; i++) {
    if (Log.isDebugEnabled())
        Log.log("PAKGetDS Output: " + dso[i], Log.DEBUG);
    try{
        setPropObject(dso[i], pakbean.getOutputDataslotValue(dso[i]));
    } catch (Exception e) {
        Log.log("Error retrieving output ds name: " + dso[i] + " error: " + e.toString(), Log.ERROR);
        throw e;
    }
}
catch(BPM WorkflowRuntimeException bizSoloEx) {
    throw bizSoloEx;
}
catch( Exception ex ) {
    mode = getPropString(PublicResources.ADAPTER_MODE);
    Log.log("PAKGetDS mode: " + mode, ex, Log.ERROR);
    setError(ex.toString());
    if (PublicResources.BIZSITE_MODE.equalsIgnoreCase(mode))
        throw ex;
    return -1;
}
return 0;
}
private String getEmailAddress(String userName) {
    if (userName == null)
        return null;
    String emailAttribute = null;
    try{
        emailAttribute = SBM.getParameter("umacl.conf", "user.email.attr");
    } catch (Throwable ignore){
    }
    if (emailAttribute == null || emailAttribute.trim().equals("")
        emailAttribute = "email";
    try{
        User user = UserManager.getUser(userName);
        return (String) user.getAttribute(emailAttribute);
    } catch (Throwable ignore){
        return null;
    }
private boolean isEmailPerformer(String perf) {
    try {
        UserManager.getUser(perf);
        return false;
    } catch (Throwable ignore) {
        return true;
    }
}

private boolean comparePerformer(String sessionUser, String performer) {
    if("true".equals(SBMConf.SBM_USERMANAGER_CASESENSITIVE)) {
        return sessionUser.equals(performer);
    } else {
        return sessionUser.equalsIgnoreCase(performer);
    }
}

Sample adapter code for BP Server-style adapters

Use the following sample Adapter code as a template to create custom BP Server-style Adapters.

Note: When compiling the java source for this adapter, please make sure to add bpmworkflow.jar and common.jar to the classpath of your compilation environment (example, JBuilder, command line interfaces, and so on).

```java
package com.savvion.BPM_webFlow.beans;
import com.tdiinc.common.AdapterInterface;
import java.util.Hashtable;
import java.util.Enumeration;
import java.lang.reflect.*;
public class DummyBP_Server_TestEP implements AdapterInterface {
    private String stringA;
    private String stringB;
    public void setAllInputDataslots(Hashtable h) throws BPMWebFlowRuntimeException {
        Field [] myFields = null;
        try {
            myFields = this.getClass().getDeclaredFields();
            AccessibleObject.setAccessible(myFields, true);
        } catch (SecurityException se) {
            // give up, log and pass the error
            se.printStackTrace();
            Log.log(se.toString());
            throw new BPMWorkflowRuntimeException(se.toString());
        }
        for (int i=0; i<myFields.length; i++) {
            Field f = myFields[i];
            Object myVar = h.get(f.getName());
            try {
                if(myVar!=null)
                    f.set(this, myVar);
            } catch (Exception ex) {
                // give up, log and pass the error
                ex.printStackTrace();
                Log.log(ex.toString());
                throw new BPMWorkflowRuntimeException(ex.toString());
            }
        }
        AccessibleObject.setAccessible(myFields, false);
    }
```
public Hashtable getAllOutputDataslots()
throws BPM WorkflowRuntimeIOException
{
    Hashtable h = new Hashtable();
    try {
        Field [] myFields = null;
        myFields = this.getClass().getDeclaredFields();
        AccessibleObject.setAccessible(myFields, true);
    } catch (SecurityException se) {
        // give up, log and pass the error
        se.printStackTrace();
        Log.log(se.toString());
        throw new BPM WorkflowRuntimeIOException(se.toString());
    }
    for (int i=0; i<myFields.length; i++) {
        Field f = myFields[i];
        try {
            Object myVar = f.get(this);
            if(myVar == null)
                myVar = "";
            h.put(f.getName(), myVar);
        } catch (Exception ex) {
            // give up, log and pass the error
            ex.printStackTrace();
            Log.log(ex.toString());
            throw new BPM WorkflowRuntimeIOException(ex.toString());
        }
    }
    AccessibleObject.setAccessible(myFields, false);
    return h;
}

public void callMe()
{
    String msg = "That's all folks :)";
    String className = this.getClass().getName();
    pr(className);
    pr(stringA);
    pr(stringB);
    pr(msg);
    Log.log(className);
    Log.log(stringA);
    Log.log(stringB);
    Log.log(msg);
}

public void setProcessContextData(Hashtable processCtx)
{
    // not required by BPM Workflow, only by BP Server
}

private void pr(String msg)
{
    System.err.println(msg);
}

Adding custom adapters to the process template

To add a customized adapter to a process template:

- After creating and storing the Adapter (class file) in the appropriate directory, enter the fully qualified Java name in the Class box of the **New Adapter** dialog box (accessed through dragging the **New Adapter** icon from the **Assign participants** link in the Tasks Pane to the process template). In this example, we use a sample class file named **MyBPM WorkflowAdapter**.

- For Adapters developed without a package description in the class file, enter the name of the class file in the Class box of the **New Adapter** dialog box.
For Adapters developed with a package description in the class file, enter in the Class box the name of the package (in this example, "packagename") and the class file name separated by a period (.).

If the Web application has been published, then save the class files in one of the following directories, where WebAppDir refers to the application server that you are using:

- `<WebAppDir>/BPM Workflow/common/classes`
- `<WebAppDir>/BPM Workflow/<Application Name>/WEB-INF/lib`

If the application has not been published already, then save the class files under the following:

- `OEBPS_HOME/BPM Workflow/common/classes`. In this case, the adapter’s class files must be copied to the AppServer directory before the application is published.
- `OEBPS_HOME/BPM Workflow/<Application Name>/WEB-INF/lib`. Here, the adapter’s class files are automatically copied to the AppServer directory when the application is published.

It is also correct to save the files to the following directory:

- `<WebAppDir>/WEB-INF/classes`

## Compiling adapters

To compile adapters, you must include `bpmwebflow.jar` file from the `OEBPS_HOME/lib` directory in the classpath (only the bizsolo.jar located under the Business Process Server directory; there is no need to include the `bpmwebflow.jar` located under the application server directory). You must also include additional *.jar files as required by the application.

After compiling an adapter, reinstall the application—if it was previously installed—to regenerate the jsp pages related to the adapter.

## Deploying adapters

Application-specific adapters are deployed in:

App_Server\ebmsapps\Application_Name\WEB-INF\classes.

Application-independent adapters are deployed in App_Server\...BPM Workflow\common\classes.

## Debugging adapters

You can use the Log class to store debugging information into the `bizsolo.log` file. The Log class implements the method `log()` that can be used to debug Adapter code. For example:

```java
Log.log("About to send email to" + bean.getPropString("userId"), Log.DEBUG);
```

The output of `Log.log()` is sent to bizsolo.log, according to Log4j specification under the `OEBPS_HOME/logs/Savvion/Business Process Server\` directory.
Using Callback Adapter

When Business Process Server change states, that event can be used to perform appropriate actions. For example, when a state change such as "BP Server suspended" occurs, an adapter can be invoked to send a pager message to the Business Process Server administrator. The Callback Adapter in Business Process Server performs this function.

By default, the Callback Adapter mechanism is disabled and no action is taken by the Business Process Server components on any change of state. The administrator has the option of enabling the Callback Adapter for both servers; or using the Callback Adapters only for the BP Server or BPM Events server without affecting the other component server.

Note: The Callback Adapter is invoked only after the server changes its state.

In the case of BP Server, callback support is also provided for state changes of process template, process instance and workstep instance. The supported states are:

- Process template — Activated, Suspended, Resumed, Removed
- Process instance — Removed, Completed
- Workstep instance — Activated, Completed

For the workstep events, you can access a reference to `com.savvion.sbm.bizlogic.server.JSTools` object from the `ProcessContext` object by calling the `getWorkflowService()` method on `ProcessContext`. You can use the `JSTools` object to access the workflow operations like getting dataslot values, and updating dataslot values.

For details, see the following topics:

- Developing the Callback Adapter
Troubleshooting the Callback Adapter

Developing the Callback Adapter

Developing and deploying a new Callback Adapter for Business Process Server involves the following steps.

1. **Understanding the XML structure** on page 104, and the XML tags.
2. **Developing the Callback Adapter Java class** on page 107, and implementing the methods to handle processing of the change of state. This processing could involve performing all workflow operations including dataslots read/write operations and notifying an external system in some proprietary manner such as sending an e-mail or pager message.
3. **Enabling the Business Process Server components to invoke the adapter** on page 109.
4. **Executing the adapter** on page 109.

The following sections describe these steps in detail.

Understanding the XML structure

A typical callback XML file looks like:

```xml
<Callbacks>
  <ServerCallbacks>
    <Location>D:\sbm\ebmsapps\common\callback</Location>
    <ServerCallback name="BPM Events">
      <OnShutdown invoke="after">
        <AdapterClass method="stop">
          com.savvion.adapters.ServerCallback2</AdapterClass>
        </OnShutdown>
      </ServerCallback>
    </ServerCallback>
    <ServerCallback name="BP Server">
      <OnSuspend invoke="after">
        <AdapterClass method="suspend">
          com.savvion.adapters.ServerCallback4</AdapterClass>
        </OnSuspend>
      </ServerCallback>
    </ServerCallback>
  </ServerCallbacks>
  <ProcessCallbacks>
    <ProcessTemplateCallback name="ptHiring">
      <Filter>
        <ProcessTemplate name="Hiring" isAppName="false" />
        <ProcessTemplate name="Approval" isAppName="true" />
      </Filter>
      <OnActivate invoke="after">
        <AdapterClass method="activate">ProcessCallback1</AdapterClass>
      </OnActivate>
    </ProcessTemplateCallback>
    <ProcessInstanceCallback name="piHiring">
      <Filter>
        <ProcessTemplate name="Hiring" isAppName="false" />  
      </Filter>
      <OnRemove invoke="after">
        <AdapterClass method="remove">ProcessCallback4</AdapterClass>
      </OnRemove>
      <OnComplete invoke="after">
        <AdapterClass method="complete">ProcessCallback4</AdapterClass>
      </OnComplete>
    </ProcessInstanceCallback>
  </ProcessCallbacks>
</Callbacks>
```
Note:
This is a sample callback configuration file. The classes mentioned in this XML file do not exist in Business Process Server installation. You need to develop the callback classes and update the XML file appropriately before using them.

The various tags in the XML file are described below.

- `<Callbacks>` — Root tag
- `<ServerCallbacks>` — Contains callback information about the servers - BP Server and BPM Events.
- `<Location>` — Provides location of the adapter class files. The default location is `OEBPS_HOME\ebmsapps\common\callback`. This tag is optional. If not included, then the default location of the class files is `OEBPS_HOME\ebmsapps`.
  
  For an application-specific callback, the default location is `OEBPS_HOME\ebmsapps\<application name>\callback`. For example, in the case of Hiring application, the default location of the callback adapter class is `OEBPS_HOME\ebmsapps\Hiring\callback`.
- `<OnStart>, <OnShutdown>, <OnSuspend>, <OnResume>`: Provide the Server state that invokes the defined classes. Attribute `invoke` defines the time of invocation, and can take values before, after or both.
- `<AdapterClass>` — Defines the class to be invoked. Attribute `method` contains the method name.
- `<ProcessCallbacks>` — Contains callback information about BP Server process (application). It contains zero or one instance of `<Location>` tag and zero or more instances of `<ProcessTemplateCallback>` and `<ProcessInstanceCallback>` tags.
- `<ProcessTemplateCallback>` — An example is shown here:

```xml
<ProcessCallbacks>
  <ProcessTemplateCallback name="ptHiring">
    <Filter>
      <ProcessTemplate name="Hiring" isAppName="false">
        <ProcessTemplate name="Approval" isAppName="true">
      </ProcessTemplate>
    </Filter>
    <OnActivate invoke="after">
      <AdapterClass method="activate">ProcessCallback1</AdapterClass>
    </OnActivate>
    <OnComplete>
      <AdapterClass method="complete">ProcessCallback4</AdapterClass>
    </OnComplete>
  </ProcessTemplateCallback>
</ProcessCallbacks>
```

OpenEdge Business Process Server: Application Developer’s Guide 105
The name attribute contains a string value for callback name, which should be unique among all callback tags.

The isAppName attribute can take values true or false. In the above example, isAppName has value as false, indicating that the name specified (in this case, 'Hiring') is that of the BP Server process template. If isAppName has value as true, then the name is taken as the generic application name, which is the common name for multiple versions of BP Server process templates.

Note that for ProcessCallback, the invoke attribute can have only one value as after.

The <ProcessTemplateCallback> tag can have maximum one instance of <Filter> tag, and maximum one instance of each of the state change tags. The possible state change tags are: <OnActivate>, <OnSuspend>, <OnResume>, and <OnRemove>.

• <Filter> — Contains information about the filter conditions that are applicable to the callback. In the above example, the Callback Adapter is registered for process template state change, which is activation of the process template. With the filter conditions specified as process template name 'Hiring', this callback is applicable only to 'Hiring' process template. Please note that there can be zero or more <ProcessTemplate> tags within the <Filter> tag.

If the <Filter> tag is omitted, then it is assumed that the callback is applicable to all process templates.

The filter conditions are kept at the state change level, within the callback tags. As multiple callback tags are supported (multiple ProcessTemplateCallback tags in this case), the user can specify different filter conditions under different callback tags.

In the above example, the activate method of the adapter class ProcessCallback1 is called when Hiring process template is activated.

• <ProcessInstanceCallback> — An example is shown here:

```
<ProcessInstanceCallback name="piHiring">
  <Filter>
    <ProcessTemplate name="Hiring" isAppName="false" />
  </Filter>
  <OnRemove invoke="after">
    <AdapterClass method="remove">ProcessCallback4</AdapterClass>
  </OnRemove>
  <OnComplete invoke="after">
    <AdapterClass method="complete">ProcessCallback4</AdapterClass>
  </OnComplete>
</ProcessInstanceCallback>
```

The name attribute contains a string value for callback name, which should be unique among all callback tags.

The <ProcessInstanceCallback> tag can have maximum one instance of <Filter> tag, and maximum one instance of each of the state change tags. The possible state change tags are: <OnRemove> and <OnComplete>.

• <WorkstepInstanceCallback> — An example is shown here:

```
<WorkstepInstanceCallback name="wiHiring">
  <Filter>
    <ProcessTemplate name="Hiring" isAppName="false">
      <Workstep>Evaluate Resume</Workstep>
      <Workstep>Feedback</Workstep>
    </ProcessTemplate>
  </Filter>
  <OnActivate>
    <AdapterClass method="activate">ProcessCallback4</AdapterClass>
  </OnActivate>
  <OnComplete>
```

The name attribute contains a string value for callback name, which should be unique among all callback tags.

The <WorkstepInstanceCallback> tag can have maximum one instance of <Filter> tag, and maximum one instance of each of the state change tags. The possible state change tags are: <OnActivate> and <OnComplete>. 
The name attribute contains a string value for callback name, which should be unique among all callback tags.

The <WorkstepInstanceCallback> tag can have maximum one instance of <Filter> tag, and maximum one instance of each of the state change tags. The possible state change tags are: <OnActivate> and <OnComplete>.

- <WorkStep> -- The workstep name for which we want to execute the callback adapter on state changes. For workstep instance call back, <ProcessTemplate> can have 0 or many <WorkStep> tags.

## Developing the Callback Adapter Java class

The Callback Adapter is a locally deployed Java class which can be loaded in the same JVM where the Business Process Server component is running. The adapter class can be in any appropriate package and need not extend any special Java class or implement any interfaces.

---

**Note:** Remote java implementations (EJB or RMI adapter classes) are not supported.

A public constructor with no arguments can be used to define the class. This constructor is used to instantiate the adapter object and then the defined methods are invoked on this adapter instance.

The adapter instance is invoked synchronously with the change of state in the Business Process Server component. A new instance of the adapter is instantiated using the default constructor and this instance is destroyed after the defined state change method is invoked. There is no pooling of object instances.

Within the adapter class, the Java language classes and APIs (System and User defined) can be used as long as they are all available in the classpath of the EJB server component.

Each adapter class is loaded by a custom defined class loader which looks for the adapter classes in a specific folder specified by the adapter developer in an adapter configuration file. If the compiled adapter class file and other related classes used by the adapter cannot be located in this folder, then they are searched in the EJB server classpath and loaded from there.

A separate adapter is not required for each Business Process Server component. The same adapter class can be used for all the components but entries must be added for each of the component in the sbmcallback.xml file in the OEBPS_HOME\conf\resources\common folder. Each of the components (BP Server, BPM Events) runs as an independent application in the EJB server, creates an adapter instance, and sends separate notifications about its state changes. The administrator has the option of using the Callback Adapters for only the BP Server or the BPM Events server, without affecting the other component. By default, the Callback Adapter mechanism is disabled and no action is taken by the Business Process Server components on any change of state.

---

**Note:**

Callback adapter executes as part of the BP Server transaction. So BP Server passes the process context to the invoked callback adapters which includes pid, wname, and other data. You can access 'JSTools' in the context so that callback adapater can perform all workflow operations, which includes dataslot read and write operations.
Implementing adapter methods

An adapter method can be specified, which is invoked by Business Process Server whenever the specific component undergoes a change of state. The current states of the components for which adapter methods can be defined are listed in the following table.

<table>
<thead>
<tr>
<th>State</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnStart</td>
<td>On start up of the Business Process Server component. For example, when BP Server or BPM Events server is started.</td>
</tr>
<tr>
<td>OnShutdown</td>
<td>On stopping of the Business Process Server component. For example, when BP Server or BPM Events server is stopped.</td>
</tr>
<tr>
<td>OnSuspend</td>
<td>On suspension of the Business Process Server component. For example, when BP Server or BPM Events server is suspended.</td>
</tr>
<tr>
<td>OnResume</td>
<td>On resuming of the Business Process Server component from suspended state. For example, when BP Server or BPM Events server is resumed.</td>
</tr>
</tbody>
</table>

The adapter developer has the option of defining a separate method to be invoked for each of the above mentioned states. Alternatively, the same method can be used to handle all the state changes. Business Process Server passes a parameter at run time to the adapter method that can be used to extract information about the actual change of state and accordingly handle it.

Using constructors

The adapter needs to have a public constructor method with no arguments. This is used to instantiate the adapter. At any instance specific resource initialization required for the adapter can be added here.

Invoking adapter methods

All methods defined to be invoked by Business Process Server should have the following syntax.

```java
public void MethodName(com.savvion.sbm.util.SBMContext c);
```

The method needs to have public access, return a void, and accept a single parameter of type `com.savvion.sbm.util.SBMContext`. This class is available in the Business Process Server ejbserver classpath.

The `SBMContext` object contains necessary information about the Business Process Server components change of state as well as additional server information about the Business Process Server instance which invoked the adapter. The callback information enables the adapter to make a callback as a Business Process Server client. You can use BP Server API or BPM Events API and have more control over the state change.

Additionally, the following classes must be imported:

- import com.savvion.sbm.util.SBMContext;
- import com.savvion.sbm.util.callback.ServerContext;
- import com.savvion.sbm.bizlogic.util.ProcessContext;
In the case of server callbacks, the object passed to the callback method is an instance of the ServerContext class. In the case of process callbacks, the object is an instance of the ProcessContext class.

Compiling and testing the adapter

After developing the adapter sources, you must build the adapter class. We recommend that you test the adapter outside the Business Process Server environment in a standalone environment. You can create a SBMContext object instance, initialize it with default values, and test all the adapter methods that Business Process Server invokes.

Enabling the Business Process Server components to invoke the adapter

Each server component includes a parameter in its conf file that specifies whether the Callback Adapter needs to be invoked or not when the state of the server component changes.

Configuring for BP Server

For enabling callback for the BP Server, use the bizlogic.callback parameter in the $OEBPS_HOME/conf/bpserver.conf file. By default, this parameter is set to false and the callback mechanism is disabled.

To enable callback for the BP Server, set the value to true, as shown in the following example:

```plaintext
bizlogic.callback = true
```

After changing the parameter’s value, restart BP Server to enable the Callback Adapter mechanism.

Configuring for BPM Events

For enabling callback for the BPM Events server, use the bizpulse.server.callback parameter in the $OEBPS_HOME/conf/bpmevents.conf file. By default, this parameter is set to false and the callback mechanism is disabled.

To enable callback for the BPM Events server, set the value to true, as shown in the following example:

```plaintext
bizpulse.server.callback = true
```

After changing the parameter’s value, restart BPM Events server to enable the Callback Adapter mechanism.

Executing the adapter

After developing the adapter class, setting up the sbmcallback.xml file and enabling the Business Process Server components callback flag to true, you must restart the Business Process Server for these changes to be applied. On restart, the Callback Adapter would be automatically invoked on the specified change of state.
Troubleshooting the Callback Adapter

On server startup, each Business Process Server component checks its own callback flag and tries to load the sbmcallback.xml file when the flag is set to true. On successful parsing of this file, it loads the adapter classes from the defined location, and validates these classes by looking for the methods defined in the xml file. If this process is successful, then these methods are registered for invocation whenever the component changes state.

During the server initialization phase, errors are logged in the OEBPS_HOME/logs/oebps.log file with detailed information regarding the error. An error message may also be displayed on the EJB server console with a detailed trace of the sequence of calls. Optionally, each Business Process Server component specific log file such as bizlogic.log may contain additional information regarding the error.

In the case of BPM Events server, when an error occurs, the server continues with its startup after printing a warning message on the EJB server console as well as the bpmevents.log file.

All exceptions generated in the adapter class must be caught by the methods and handled appropriately. If these are not caught by the adapter method and propagated to Business Process Server, then an appropriate error message is displayed on the EJB server console and the message is written either to the Business Process Server components log file (bserver.log or bpmevents.log), or the oebps.log file. The Business Process Server continues its processing after logging this exception.
Developing applications with rollback support

In certain situations, it might be required to roll back to a workstep which was already completed earlier in the process. To accomplish this, you restart the workflow from the desired rollback workstep in the process. Numerous state changes can occur to the process between the selected rollback workstep and the current activated workstep. When the rollback workstep is reactivated, the process goes back to the same state as when the rollback workstep was first activated. This compensatory enabled process is necessary to restore the original information and eliminate actions done by the performers in the worksteps prior to the rollback process. These actions are achieved using JavaScript in the completed worksteps to undo the changes to the state of the process.

In addition to describing how to reload the data using the compensatory enabled process, this chapter covers:

- The definition and limitations of the compensatory enabled process.
- A sample process illustrating how rollback points, data slot selection, compensatory JavaScript, and re activate worksteps are set up in the process template file.

**Note:** Additional information regarding defining worksteps in the BPM Designer interface are explained in the *OpenEdge Getting Started: Developing BPM Applications with Developer Studio*.

For details, see the following topics:

- Attributes of the compensatory enabled process
- Rollback restrictions
- Rollback process example
Writing JavaScript code for rollback worksteps

Attributes of the compensatory enabled process

The primary attributes of the compensatory enabled process are rollback point and dataslot selection. However, dataslot selection is an optional attribute. The other optional attributes are compensatory JavaScript and activate rollback points.

Once you set these attributes, execution of the rollback occurs when you set the rollback point so that if the current workstep fails under execution, then Business Process Server internally rollbacks to a specified point defined in this workstep.

To prepare the compensatory enabled process for activation, set the mandatory and optional attributes covered in the following sections.

Creating rollback points

You may decide actions regarding each particular workstep and the related workitems and where to set a rollback point for a workstep. However, there are restrictions on selecting rollback points, as described in Rollback restrictions on page 115.

To set a rollback point:

1. From the BPM Designer Tasks pane, click the Connect Shapes link.
2. Click the Draw compensation flow connector link
3. Connect the source workstep to the workstep to which you want to rollback.
   
   BP Server sets the rollback point for this workstep.

   Note: BP Server does not expose any API to set or reset this attribute.

Selecting dataslots for replacement

Some processes use many dataslots and replacing all dataslots in each rollback may compromise server performance. For this reason, you may specify dataslot names for replacement. This option helps the workflow process to replace the chosen dataslot values with the original dataslot values.

To select dataslots for replacement, complete the following:
1. Complete steps 1-3 of the procedure *Creating rollback points* on page 112.

2. From the **Properties** view of the rollback workstep, click the **Advanced** tab, and then the **On Activation** tab.

**Figure 19: Setting the rollback point**

3. Click the **ellipsis** button beside the **Save dataslots values for:** box, to open the **Save Dataslots Values** dialog box.

4. Click **Add** and then select a dataslot from the **Select Dataslots** dialog box. Click **OK** to add the dataslot to the **Name** list.

5. Repeat Step 3 for additional dataslots, if necessary.

6. Click **OK**.

BP Server sets these selected dataslots for replacement.

**Note:** The first time a rollback workstep is activated, BP Server takes a snapshot of all selected dataslot values and persists it for later replacement. If the workstep defined as a rollback point is skipped due to a skip condition, then the dataslot values for that workstep are not persisted. Therefore, when BP Server rolls back to a skipped workstep, the dataslot values of such a workstep are not replaced with its original values.

**Writing compensatory JavaScript code (optional)**

You can use the compensatory enable process to write compensatory JavaScript codes to handle a rollback request. For example, write JavaScript code in a process template using rollback points for the following cases:

- Send messages or e-mail notifications before running pre/post-functions for a second time or for undoing process effects regarding these functions.
- Send e-mail notifications to a computer where an asynchronous nested process is running.
- Actions applied for rollback to a loop.

Compensatory JavaScript also works as pre- and post-functions for worksteps and cleans all activities completed by this workstep, such as deleting generated files or removing new database entries.

To write the compensatory JavaScript code, complete the following steps:
1. Click the workstep to which you want to rollback.

2. From the Properties view, click the Advanced tab, and then the On Recovery tab.

3. Write your code in the Execute script text area. BP Server sets the compensatory JavaScript for this workstep.

In BP Server, if you write the compensatory JavaScript, then it is compiled and inserted automatically into the XML file, similar to how pre- and post-functions are compiled. You can use compensatory JavaScript in any workstep, whether or not Create Rollback Point is selected for the workstep. The compensatory JavaScript is executed only when a rollback is requested by you or by the BP Server through a workstep’s execution failure (described in the next section).

Activating rollback points (optional)

As described in the previous section, BP Server activates an internal call to rollback when there is an execution failure for:

- Post function scripts, or
- Adapter or subprocess (nested) worksteps.

**Note:** BP Server no longer supports activating Rollback points in a workstep, for an exception failure in pre-function scripts.

By specifying a workstep name, every execution failure causes a rollback to the given workstep name.

**To specify a workstep, complete the following steps:**

1. Open the Properties view of the workstep, which may fail.
2. From the Advanced tab, click the On Error tab. Select a workstep from the Activate rollback point list.

**Note:** If the On error area is not active, then ensure that the Create rollback point checkbox is selected in the workstep you want to roll back to.

When you request rollback to a workstep, the BP Server ensures that:

- The workstep has been completed and
- The Create rollback point checkbox is selected.

If these two conditions are satisfied, then BP Server:

1. Suspend the current process instance (no data is exchanged).
2. Suspend all active worksteps.
3. Run the compensatory JavaScript codes for all worksteps that were visited.
4. Replace all given dataslot values with default values at that point.
5. Resume the current process instance.
6. Activate the given workstep.
7. Send an e-mail to the manager of the process, if you have requested it.
Note: If the two above conditions are not met, then the user remains in the same workstep. There is no change in the process data and BP Server throws an exception.

Rollback restrictions

Rollback has the following restrictions:

- Apply rollback points only for Activity, Adapter, and Subprocess worksteps. BP Server does not allow rollback to And gateway, Or-join, Decision, Split, Start, and End worksteps.

- Rollback points are not allowed between a Split and And gateway/Or-join in the process template. Placing a rollback point between these types of connectors causes errors.

- We do not recommend applying a rollback to inline Subprocesses.

- Asynchronized Subprocess and Adapter worksteps run in separated threads. When placing rollback points in these worksteps (in addition to worksteps placed before asynchronized Subprocess and Adapter worksteps), BP Server must stop these kinds of threads. You can remove the created subprocess instance threads. However, removing Adapter workstep threads is not possible. The only solution is an e-mail notification.

- You cannot create a backup of system dataslots for the purpose of rollback.
The outlined area (labeled "Not a Safe Point") in the Figure 20: Areas in a process template where rollback points are not allowed on page 116, displays areas in the process template where rollback functionality is not allowed. Both, the rollback start point (the workstep that is configured to rollback to another workstep on error) and the rollback end point (the workstep to which the rollback happens), must be outside the outlined area. If any one or both these rollback points are inside the outlined area, then it leads to unexpected errors.

Figure 20: Areas in a process template where rollback points are not allowed
Rollback process example

Let us consider the following process template as an example of how to create and define a process template using the rollback process. Figure 21: Sample process using a rollback functionality on page 117 represents a Purchase Request process template.

Figure 21: Sample process using a rollback functionality

![Diagram of the process template](image)

The Purchase Request process contains the following user-defined dataslots.

<table>
<thead>
<tr>
<th>Dataslot name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product_ID</td>
<td>CHARACTER</td>
<td>ID of the requested product</td>
</tr>
<tr>
<td>Product_Name</td>
<td>CHARACTER</td>
<td>Name of the requested product</td>
</tr>
<tr>
<td>Product_Description</td>
<td>CHARACTER</td>
<td>Description of the requested product</td>
</tr>
<tr>
<td>Product_Quantity</td>
<td>INTEGER</td>
<td>Quantity of the requested product</td>
</tr>
<tr>
<td>Product_Price</td>
<td>INTEGER</td>
<td>Price of the requested product</td>
</tr>
<tr>
<td>Estimated_DeliveryTime</td>
<td>DATETIME-TZ</td>
<td>Estimated date of product delivery</td>
</tr>
<tr>
<td>approved</td>
<td>LOGICAL</td>
<td>Indicates if the order is approved or not</td>
</tr>
</tbody>
</table>

The process is summarized as follows: An employee requests an item. If the manager approves the item, then a purchase order program is executed. In addition to searching for the availability of the requested item, the program can save the information in a database to track the employee order and update entries in the database. In the final workstep, "Order Approval," a manager approves the order and the order is placed.

You may want rollback functionality in the process, when the following situations occur:

- The adapter operation for the "Purchase Order" workstep fails.
The person performing the "Order Approval" workstep does not approve this request.

Table 13: Sample application workstep properties on page 118 summarizes the workstep properties in this example.

### Table 13: Sample application workstep properties

<table>
<thead>
<tr>
<th>Workstep</th>
<th>Rollback point</th>
<th>Dataslots to be replaced</th>
<th>Reactivate workstep</th>
<th>Compensatory JavaScript</th>
</tr>
</thead>
<tbody>
<tr>
<td>Submitter</td>
<td>Not set</td>
<td>None</td>
<td>Not set</td>
<td>None</td>
</tr>
<tr>
<td>Manager Approve</td>
<td>Set</td>
<td>Product_ID, Product_Description, Product_Quantity, Product_Price, Product_Name, Estimated_DeliveryTime, approved</td>
<td>Not set</td>
<td>None</td>
</tr>
<tr>
<td>Purchase Order</td>
<td>Not set</td>
<td>None</td>
<td>Manager Approve</td>
<td>Packages. PurchaseRequest. externalperformers.Order. clean()</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Order Approval</td>
<td>Not set</td>
<td>None</td>
<td>Manager Approve</td>
<td>None</td>
</tr>
</tbody>
</table>

The "Manager Approve" workstep has a rollback point set to allow rollback to this workstep. The dataslot values of Product_Description, Product_ID, Product_Quantity, Product_Price, Product_Name, and Estimated_DeliveryTime can be replaced at a later stage in the process with the dataslot values from this workstep.

The Decision gateway has two outgoing links, "Approved" and "Not Approved" (default). For the "Approved" conditional link, configure the condition with "approved" dataslot value set to true.

The "Purchase Order" workstep also includes compensatory JavaScript, which can be added to reactivate the "Manager Approve" workstep and clean the database of changes the failed adapter might have made.
To define the compensatory JavaScript in the **On Recovery** area under the **Advanced** tab, click in the **Execute script** text area and make changes or add new script as required (Figure 22: Compensatory JavaScript set for on recovery on page 119).

**Figure 22: Compensatory JavaScript set for on recovery**

---

### Writing JavaScript code for rollback worksteps

The following restrictions apply to using JavaScript for rollback worksteps:

- You write your JavaScript code as functions.
- Do not dangle any expressions outside the functions.
- Use only one driver function, which can invoke other functions.
- Third-party tools in JavaScript must be used as static methods.

**Note:** You cannot debug JavaScript using a debugger. However, you can use the jst.writeLog() method to print the debug information in the log file. The JavaScript Engine integration code writes the information or error messages to the bizlogic.log file.

---

### Sample Javascript code

In the following example, "jst" refers to JSTools class for execution of JavaScript codes. Users can invoke all getXXX() and setXXX() methods and also specialized tools such as sendEmail().

```javascript
var name = jst.getDataSlotValue("empName");
var salary = jst.getDataSlotValue("empSalary");
jst.putDataSlot("empName", name);
jst.putDataSlot("empSalary", salary);
var adr = "ebms@savvion.com";
var subject = "Testing";
var content = "CONTENT";
jst.sendEmail(adr, subject, content);
jst.writeLog(Done);
```

In the following example, "pak" refers to the entire BP Server API. All API calls must start invoking pak.connect(<username>, <password>) and end with pak.disConnect().
Example:

```javascript
var session = pak.connect(<username>, <password>);
var ebmsVersion = pak.getPAKVersion(session);
pak.disConnect(session);
```

In the following example, the keyword "Packages" invokes static Java methods.

Example:

```
Employee.java
package com.tdiinc;
public class Employee {
    protected static String empName;
    public static String getName () {
        return empName;
    }
}
```

```
JavaScript code
function f1() {
    var name = Packages.com.tdiinc.Employee.getName();
    return (name);
} f1()
```
Using the Rule Wizard framework

The Rule Wizard Framework facilitates building the wizards that generate rule files for solving a specific business problem. A wizard is a sequence of pages which guides users through defining a rule while they enter data. After data is entered, a rule module addressing the specific business problem is generated. To define a wizard, you must define the data structures required to hold the parameters, the GUI, and the code generation—all contained in a rule template in XML format.

When deciding to build a rule wizard for a specific task, consider the following input components:

- The business problem to solve along with its solution using rules.
- The parameters leading to variations in the rules module which solves the problem.
- The GUI requirements which allow the user to specify values for the parameters mentioned in the above point.

The rule wizard Framework facilitates wizard building by:

- Making possible the building of the wizard GUI without any Java code in some cases or by designing simple components in other cases.
- Providing a powerful macro language (Velocity) in which the rule template writer can express variations of the rules according to the parameters.

After defining the business problem and its solution using parameterized rules, the rule template writer completes the following tasks:

1. Design the data structures or macros holding the parameters.
2. Design the GUI.
3. Write the code generation portion.
The following example illustrates this process without generating valid rule language code, but uses only the data entered by the user. The rule template follows, using the simplest data structure available—a string:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE template SYSTEM "resources/ruleTemplate.dtd">
<template name="ex1">
  <screen number="10">
    <editor macro="text" label="Text"/>
  </screen>
  <macro name="text" type="varname"/>
  <text>Hello, you entered $text.</text>
</template>
```

Now run the Rule Wizard using this rule template as input:

```
RuleWizard -t ex1.rtu
```

where `-t` option is used for specifying the RuleTemplate file name.

The `RuleWizard.cmd` file is located in the `OEBPS_HOME\bin` directory. Assume that the `OEBPS_HOME\bin` directory is in your PATH and that the sample code is saved in the file `ex1.rtu`. The extension `.rtu` stands for "rule template unit."

The `<template>` tag describes the rule template unit. The `<screen>` tag represents a step in the wizard. The `<macro>` tag is the only data structure used in this rule template. Rule Wizard provides some predefined macro types. A macro type defines the data type and also specifies a default editor. For example, the type `varname` specifies the need for a string conforming to the variable name restriction in the Java language. The editor is a simple JTextField.

The `<editor>` tag appears in the first step of the wizard, as it is contained on the first screen of the template. A label is specified—in the example, “Text”—and displayed beside the editor requesting the user to enter the required information.

The Text box appears when running the wizard, as shown in Figure 23: Rule wizard example 1 on page 122.

**Figure 23: Rule wizard example 1**
Click **Finish** to verify the `varname` type of the macro you specified as well as the validations performed. If you type a legal value, such as `Example` and click **Finish** again, then the following message appears on the console:

```
Hello, you entered Example.
```

Notice that, in the code generation portion within the `<text>` tag, you can retrieve the value of the `text` macro by writing `$text`.

To add more steps to the wizard, repeat the process by duplicating the tags shown in the first example. Macro names must be unique in a template, editors must be associated to a macro, and pages are ordered according to the values in their `number` attribute.

In the case of Schedule Wizard, you can specify the application name while invoking RuleWizard.

For example,

```
%OEBPS_HOME%\bin>RuleWizard
-t %OEBPS_HOME\templates\Schedule.rtu -a Assignment
-e %OEBPS_HOME\ebmsapps\%app%\rules\SC_instancename_param.rtp.
```

where,

- `-t` option is used for specifying RuleTemplate file name.
- `-a` option is used for specifying application name.
- `-e` option is used for specifying the ruletemplate parameter file name.

For details, see the following topics:

- Composing macros and editors
- Controlling the layout
- Specifying alternatives
- Collecting macros
- Using editors in separate dialogs
- Code generation
- Standard macro types
- How to write additional macros and editors
- Skipping a macro

### Composing macros and editors

In order to define more complex GUIs and data structures, you can define macros that contain macros and editors that contain editors. An example follows:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE template SYSTEM "resources/ruleTemplate.dtd">
<template name="ex2">
<screen number="10">
<editor macro="aggregate">
<editor macro="text" label="Text">
<parameter name="columns" value="30"/>
```

OpenEdge Business Process Server: Application Developer's Guide
Chapter 10: Using the Rule Wizard framework

```xml
<editor macro="list" label="List"/>
</editor>

<macro name="aggregate" type="composite">
<macro name="text" type="varname"/>
<macro name="list" type="combo-box">
<vector-property name="listLabels">
<vector-elem>Option A</vector-elem>
<vector-elem>Option B</vector-elem>
<vector-elem>Option C</vector-elem>
</vector-property>
<vector-property name="listValues">
<vector-elem>option_a</vector-elem>
<vector-elem>option_b</vector-elem>
<vector-elem>option_c</vector-elem>
</vector-property>
<property name="listSelectedValue">option_b</property>
</macro>
</macro>

Value of text is $aggregate.getMacro("text").
Value of list is $aggregate.getMacro("list").selectedValue.

Run as follows:

RuleWizard -t ex2.rtu

The List box is now added, as shown in Figure 24: Rule wizard example 2 on page 124.

Figure 24: Rule wizard example 2

In this example, the composite macro contains two components: one is the varname macro, also used in the first example, and the combo-box macro.

The combo-box macro provide users with a list of possible values allowing them to choose. The listLabels vector property specifies the display names of the values and the actual values are specified in the listValues property. The listSelectedValue property is used to display the selected default value. Notice the new notion of a macro property. These properties are of type string, vector of strings, and hashtable containing strings as keys and values.
Controlling the layout

In this example, the <editor> tag contains other editors. The top editor is associated with the macro of type composite, with the name aggregate. Simple, non-qualified names are used to specify the macros for the editors inside, since associating the top editor with the macro aggregate makes the component editors search for their macros within this macro. The composite editor is able to generate an automatic layout for the component editor.

The size of the JTextField control has changed due to use of the <parameter> tag inside the <editor> tag.

Type Example2 in the text field, select Option C in the list, and click Finish. The following message appears on the console:

Value of text is Example2.
Value of list is option_c.

Notice how the macros within the aggregate macro are addressed in the code generation part of the rule template:

$aggregate.getMacro(list).selectedValue

In fact, you can invoke the getMacro method with the argument list, which is the component macro name. You can then request the value of the selectedValue property.

Controlling the layout

The second example uses the automatic layout of editor components provided by the Rule Wizard. To customize the layout, you can use the <constraints> and <label-constraints> tags within the <editor> tags. This leads to the third example:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE template SYSTEM "resources/ruleTemplate.dtd" >
<template name="ex3">
<screen number="10">
  <editor macro="aggregate">
    <editor macro="text" label="Text">
      <parameter name="columns" value="30"/>
      <label-constraints fill="HORIZONTAL" insets="2,2,2,2" weightx="1" gridx="0" gridy="0"/>
      <constraints fill="NONE" insets="2,2,2,2" weightx="2" gridx="1" gridy="0" anchor="WEST"/>
    </editor>
    <editor macro="list" label="List">
      <label-constraints fill="HORIZONTAL" insets="2,2,2,2" weightx="1" gridx="0" gridy="1"/>
      <constraints fill="NONE" insets="2,2,2,2" weightx="2" gridx="1" gridy="1" anchor="WEST"/>
    </editor>
  </editor>
</screen>
</template>
```

OpenEdge Business Process Server: Application Developer's Guide 125
Notice how the controls are aligned to the left, as shown in Figure 25: Rule wizard example 3 on page 126.

Figure 25: Rule wizard example 3

The composite editor uses a GridBagLayout. You can specify the GridBagConstraints for each label and for each editor using the <constraints> and <label-constraints> tags. Any property you set in a GridBagConstraints object can be specified as an attribute of one of these tags. Introduce constants defined by the GridBagConstraints class to use their symbolic name (such as HORIZONTAL).

Specifying alternatives

A frequently occurring situation is when two macros exist but only one macro may contain valid values at a time. The user must decide which macro contains the valid values. The macro of type composite provide the groupComponents and selectedMacro properties. The first property contains the names of the component macros which belong to this group (this means only one contains valid values) and the latter (selectedMacro) contains the default value (which component is valid when the components first display). Besides components belonging to a group, a composite macro may contain components that are valid at all times. Slightly modifying the third examples show the following:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE template SYSTEM "resources/ruleTemplate.dtd" >
<template name="ex4">
<editor macro="aggregate">
    <editor macro="text" label="Text">
        <parameter name="columns" value="30"/>
        <label-constraints fill="HORIZONTAL" insets="2,2,2,2" weightx="1" gridx="0" gridy="0"/>
        <constraints fill="NONE" insets="2,2,2,2" weightx="2"
```
Specifying alternatives

Now the boxes appear as option buttons, as shown in Figure 26: Rule wizard example 4 on page 127.

Figure 26: Rule wizard example 4
Notice that validation occurs only on the selected macro. For example, select an option in the List box while leaving the Text box empty. Notice that the wizard does not display any error message when you click Finish.

**Note:** Watch for the if construct in the Velocity language used in the code generation portion in order to display the macro chosen by the user.

### Collecting macros

Another frequent requirement encountered when building rule templates is the collection of macros. To support this, the <macro-type> tag defines a new type of macro with pre-existing components. An example follows:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE template SYSTEM "resources/ruleTemplate.dtd" >
<template name="ex5">
  <macro-type name="my-collection"
    class="com.savvion.rulewizard.data.AggregateData">
    <macro-component name="list" label="List"
      editor="editor_list"/>
    <macro-component name="text" label="Text" editor="editor_text"/>
  </macro-type>

  <editor name="editor_list" macro="list" label="List"/>
  <editor name="editor_text" macro="text" label="Text"/>

  <screen number="10">
    <editor class="com.savvion.rulewizard.gui.AggregateEditor"
      label="Heterogenous collection" macro="heterogenous"
      show-label="false">
      <constraints fill="BOTH" insets="5,5,5,5" weightx="2" gridx="0"
        gridy="0" anchor="CENTER" weighty="1"/>
    </editor>
  </screen>

  <macro name="list" type="combo-box">
    <vector-property name="listLabels">
      <vector-elem>Option A</vector-elem>
      <vector-elem>Option B</vector-elem>
      <vector-elem>Option C</vector-elem>
    </vector-property>
    <vector-property name="listValues">
      <vector-elem>option_a</vector-elem>
      <vector-elem>option_b</vector-elem>
      <vector-elem>option_c</vector-elem>
    </vector-property>
    <property name="listSelectedValue">option_b</property>
  </macro>

  <macro name="text" type="varname"/>
  <macro name="heterogenous" type="my-collection"/>

  <text>
    #foreach($single_macro in ${heterogenous.getComponents()})
    #if ($single_macro.type == "varname")
      Value of text is $single_macro.
    #else
      Value of list is $single_macro.selectedValue.
    #end
    #end
  </text>
</template>
```
The `<macro-type>` tag defines a heterogeneous collection of macros. Each component is described in a `<macro-component>` tag containing references to macro names (text and list) and editor names (editor_text and editor_list). The macros and editors are defined as ordinary, except that editors have a supplementary name attribute.

This rule template unit generates a drop-down list, as shown in Figure 27: Rule wizard example 5 on page 129.

**Figure 27: Rule wizard example 5**

With this wizard, the user can add list and text macros to the collection. Dialog boxes appear allowing the user to enter values for these macros. The standard behavior allows the user to add values no matter how many components exist in the collection. Write a Java class derived from `com.savvion.rulewizard.data.AggregateData` when another policy is required. All the references passed to a collection macro are passed to its components.

After adding a few components of different types to the collection, click **Finish** to view the results. Notice the `foreach` Velocity construct used to display the results. Iterate through the collection obtained by calling the collection macro’s `getComponents` method. Because collection components contain different types, find a component’s type by accessing the `type` property. Based on this type, you decide what properties to use.

---

**Using editors in separate dialogs**

Whenever you want to save space on a wizard page (because many other editors show), use a proxy editor to display the macro value but not all the GUI components required for editing. Alternatively, you can do the editing in a separate window, by modifying the fourth example, as illustrated in the following example. Use the named editor, `aggregate_editor`, in order to pass it as a parameter (editorName) to the proxy editor implemented by the `com.savvion.rulewizard.gui.FieldEditor`. The macro, editor definition, and code definition portion remain unchanged. The proxy editor contains its own label and also a `columns` parameter.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE template SYSTEM "resources/ruleTemplate.dtd" >
<template name="ex6">
<editor name="aggregate_editor" macro="aggregate">
    <editor macro="text" label="Text">
        <parameter name="columns" value="30"/>
    </editor>
</editor>
</template>
```
The template generates the following screen (left image, Figure 28: Rule wizard example 6 on page 130). Click the ellipsis button to open the dialog box (right image, Figure 28: Rule wizard example 6 on page 130).

Figure 28: Rule wizard example 6

Code generation

The code generation phase is written in the Velocity language. All text contained within the <text> tag is treated as a Velocity program and evaluated after the last step of the wizard. All macro objects are available to the Velocity program and contain valid data. You can refer to their properties and invoke their methods using specific Velocity syntax. The actual rule file constitutes the output of this program.

For additional information about Velocity, including a user guide and reference guide, visit the http://velocity.apache.org/.
Standard macro types

This section discusses the standard macros and their editors, properties, methods, parameters, references, and arguments.

- A macro **property** is a value used during the code generation phase and is accessed by an expression such as $macro.property.

- An **input property** is part of the rule template that specifies configuration information or default values. One specifies values for input properties by using the property, vector-property, and hash-property inside the <macro> tag.

- A macro **method** is invoked during the code generation phase and is accessed by $macro.method (list of arguments).

- A **reference** is described in a <reference> tag and contained in a <macro> or <editor> tag. References work similar to named parameters to a function. They also contain names known to the editor or macro and values which are expressions of type macro.property and evaluated at run-time. For references contained in editors, only macro names are valid as values (properties are not invoked). For references contained in macros, only one level of properties are valid as values, while an expression such as macro.property1.property2 and method calls are not allowed.

- An **argument** is described in the <argument> tag and contained in the <macro> tag. The Rule Wizard retrieves an array of arguments from BPM Designer. Each element contains a process template description and an array of selected worksteps in that process template. The <argument> tag allows for specifying indexes in these arrays so a macro knows where to obtain the data.

- An **editor parameter** is described in the <parameter> tag and contained in the <editor> tag. This parameter has the name and value attributes (plain strings) which are passed to the editor. Use these values to indicate some customization specific to the editor.

Properties common to all macros

Table 14: Properties common to all macros on page 131 provides a description of the properties common to all macros:

Table 14: Properties common to all macros

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>defaultProperty</td>
<td>A special property accessed simply by writing the macro name ($macro). If applicable, then returns the macro value and the data that the user entered as a string. Usually returns the same value as the stringValue property.</td>
</tr>
<tr>
<td>name</td>
<td>The macro name.</td>
</tr>
<tr>
<td>type</td>
<td>The macro type.</td>
</tr>
<tr>
<td>className</td>
<td>The fully qualified class name implementing the macro.</td>
</tr>
<tr>
<td>label</td>
<td>The macro label appearing on the GUI.</td>
</tr>
</tbody>
</table>
The user description of the macro and the data inside. Explains in natural language the data significance.

The same as long label but in html format.

If applicable, then returns the macro value and the data that the user entered as a string.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>longLabel</td>
<td>The user description of the macro and the data inside. Explains in natural language the data significance.</td>
</tr>
<tr>
<td>htmlLabel</td>
<td>The same as long label but in html format.</td>
</tr>
<tr>
<td>stringValue</td>
<td>If applicable, then returns the macro value and the data that the user entered as a string.</td>
</tr>
</tbody>
</table>

The next section provides the list of standard macro types. Each macro type contains a short description.

**Macro type: varname**

Allows the user to enter a string that follows the Java language restrictions for variable names.

**Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>defaultProperty</td>
<td>The string entered by the user.</td>
</tr>
<tr>
<td>stringValue</td>
<td>The string entered by the user.</td>
</tr>
</tbody>
</table>

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>columns</td>
<td>The number of columns the JTextField should contain.</td>
</tr>
</tbody>
</table>

**Macro type: boolean**

Allows the user to enter a boolean value in a checkbox.

**Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>defaultProperty</td>
<td>true or false as strings.</td>
</tr>
<tr>
<td>stringValue</td>
<td>true or false as strings.</td>
</tr>
<tr>
<td>boolVal</td>
<td>true or false as booleans.</td>
</tr>
</tbody>
</table>
## Input properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>string value</td>
<td>true or false as strings.</td>
</tr>
</tbody>
</table>

### Macro type: date

Allows the user to enter a date.

### Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>defaultProperty</td>
<td>The date the user entered and expressed as a valid date in the rule language: (YEAR:2000/MONTH:12/DAY:31/HOUR:12/MIN:00/SEC:00).</td>
</tr>
<tr>
<td>stringValue</td>
<td>The date the user entered and expressed as a valid date in the rule language: (YEAR:2000/MONTH:12/DAY:31/HOUR:12/MIN:00/SEC:00).</td>
</tr>
</tbody>
</table>

### Input properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>year</td>
<td>The components of the date value.</td>
</tr>
<tr>
<td>month</td>
<td></td>
</tr>
<tr>
<td>day</td>
<td></td>
</tr>
<tr>
<td>hour</td>
<td></td>
</tr>
<tr>
<td>minute</td>
<td></td>
</tr>
<tr>
<td>second</td>
<td></td>
</tr>
</tbody>
</table>

### Macro type: date-expression

Allows the user to enter a date or a duration.
Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>defaultProperty</td>
<td>A time interval expressed as either:</td>
</tr>
<tr>
<td></td>
<td>• A valid rule language expression</td>
</tr>
<tr>
<td></td>
<td>(YEAR:2000/MONTH:12/DAY:31/HOUR:12/MIN:00/SEC:00) if the user entered</td>
</tr>
<tr>
<td></td>
<td>a date.</td>
</tr>
<tr>
<td></td>
<td>• NOW + 3*DAYS if the user entered a 3 day duration.</td>
</tr>
<tr>
<td>stringValue</td>
<td>A time interval expressed as either:</td>
</tr>
<tr>
<td></td>
<td>• A valid rule language expression</td>
</tr>
<tr>
<td></td>
<td>(YEAR:2000/MONTH:12/DAY:31/HOUR:12/MIN:00/SEC:00) if the user entered</td>
</tr>
<tr>
<td></td>
<td>a date.</td>
</tr>
<tr>
<td></td>
<td>• NOW + 3*DAYS if the user entered a 3 day duration.</td>
</tr>
</tbody>
</table>

Input properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>timeOutExpressedAs</td>
<td>Valid values are date and duration according to what the user entered as an absolute date value or a duration.</td>
</tr>
<tr>
<td>businessTime</td>
<td>Valid values are true and false. According to these values, the duration is measured as business time or ordinary time.</td>
</tr>
<tr>
<td>durationArray</td>
<td>A vector property containing the values which define the duration: durationArray [0]= no of days;durationArray [1]= no of hours;durationArray [2]= no of min;durationArray [3]= no of sec. The value -1 is a placeholder for an invalid value.</td>
</tr>
<tr>
<td>year</td>
<td>The components of the date value.</td>
</tr>
<tr>
<td>month</td>
<td></td>
</tr>
<tr>
<td>day</td>
<td></td>
</tr>
<tr>
<td>hour</td>
<td></td>
</tr>
<tr>
<td>minute</td>
<td></td>
</tr>
<tr>
<td>second</td>
<td></td>
</tr>
</tbody>
</table>

Macro type: combo-box

Allows the user to choose a value from a combo box.
### Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>defaultProperty</td>
<td>The selected value.</td>
</tr>
<tr>
<td>stringValue</td>
<td>The selected value.</td>
</tr>
<tr>
<td>selectedValue</td>
<td>The selected value.</td>
</tr>
</tbody>
</table>

### Input properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>listLabels</td>
<td>A vector property containing the labels for the GUI component display.</td>
</tr>
<tr>
<td>listValues</td>
<td>A vector property containing the values corresponding to the values in listValues.</td>
</tr>
<tr>
<td>listSelectedValue</td>
<td>The value selected by default.</td>
</tr>
</tbody>
</table>

### Macro type: list-box

Allows the user to choose a value from a list box.

### Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>defaultProperty</td>
<td>The selected value.</td>
</tr>
<tr>
<td>stringValue</td>
<td>The selected value.</td>
</tr>
<tr>
<td>selectedValue</td>
<td>The selected value.</td>
</tr>
</tbody>
</table>

### Input properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>listLabels</td>
<td>A vector property containing the labels for the GUI component display.</td>
</tr>
<tr>
<td>listValues</td>
<td>A vector property containing the values corresponding to the values in listValues.</td>
</tr>
<tr>
<td>listSelectedValue</td>
<td>The value selected by default.</td>
</tr>
</tbody>
</table>
Macro type: enumeration

Allows the user to build a vector of strings.

**Methods**

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>getEnumeration()</td>
<td>The vector containing the strings the user entered.</td>
</tr>
<tr>
<td>getSize()</td>
<td>The size of the vector returned by getEnumeration().</td>
</tr>
</tbody>
</table>

**Input properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>enumeration</td>
<td>A vector property containing the initial values of the vector of strings.</td>
</tr>
</tbody>
</table>

Macro type: scale

Allow the user to set the limits of an interval (2 integers) and a step used to define sub-intervals.

**Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>The start of the interval.</td>
</tr>
<tr>
<td>end</td>
<td>The end of interval.</td>
</tr>
<tr>
<td>step</td>
<td>The step.</td>
</tr>
<tr>
<td>intervalNumber</td>
<td>The number of sub-intervals found between start and end using the specified step.</td>
</tr>
<tr>
<td>intervals</td>
<td>A vector of string labels for sub-intervals (for example, 2..4 for a sub-interval from 2 to 4).</td>
</tr>
</tbody>
</table>

**Input properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>start</td>
<td>The default values for the three properties.</td>
</tr>
</tbody>
</table>
Macro type: dummy
Displays a label and not a full-fledged editor. Contains only standard properties.

Macro type: label
Displays text that provides feedback to the user concerning the process template and the workstep to which he/she applies the rule template.

References

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>workstep</td>
<td>An expression of type macro.property which gives the workstep name the rule template is applied to. If the expression evaluates to null, then the user has not yet chosen a workstep. If evaluated to the empty string, then the rule template applies to the entire process.</td>
</tr>
<tr>
<td>processTemplateName</td>
<td>An expression of type macro.property which gives the process template name the rule template is applied to. If the expression evaluates to null, then the user has not yet chosen a process template.</td>
</tr>
</tbody>
</table>

Macro type: description
Displays the HTML-formatted text.

Input properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>stringValue</td>
<td>The text for display.</td>
</tr>
</tbody>
</table>

Editor parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>width</td>
<td>The width in pixels of the area displaying the text.</td>
</tr>
<tr>
<td>height</td>
<td>The height in pixels of the area displaying the text.</td>
</tr>
</tbody>
</table>

Macro type: dynamic-description
Displays the HTML formatted text obtained from evaluating a Velocity program. This program may use all the defined macros (usually the htmlLabel property of the macros) in order to build a natural language description of the data entered by the user.
### Input properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>stringValue</td>
<td>The velocity program evaluated in order to obtain the text for display.</td>
</tr>
</tbody>
</table>

### Editor parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>width</td>
<td>The width in pixels of the area displaying the text.</td>
</tr>
<tr>
<td>height</td>
<td>The height in pixels of the area displaying the text.</td>
</tr>
</tbody>
</table>

### Macro type: composite

Provides a macro that may contain other macros. The associated editor lays out multiple component editors. From the macro components belonging to a group, only one contains valid values at any moment in time.

### Input properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>groupComponents</td>
<td>A vector property containing the macro names belonging to the group.</td>
</tr>
<tr>
<td>selectedMacro</td>
<td>The name of the selected macro.</td>
</tr>
</tbody>
</table>

### Macro type: process

Allows the user to choose a process template file or receive the process template data from arguments. If the macro receives the process template data from BPM Designer, then the macro is not displayed on the GUI.

### Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>defaultProperty</td>
<td>N/A</td>
</tr>
<tr>
<td>stringValue</td>
<td>N/A</td>
</tr>
<tr>
<td>processTemplateName</td>
<td>The process template name.</td>
</tr>
<tr>
<td>processTemplateFilename</td>
<td>The process template file name (null if the process template was not saved in BPM Studio).</td>
</tr>
</tbody>
</table>
Arguments

The **processIndex** attribute of the `<argument>` tag contains the index of the ProcessArgs object in the arguments array.

**Macro type: workstep**

Allows the user to choose a workstep from a list or receive the selected workstep from arguments. If the macro receives the process template data from BPM Designer, then the macro is not displayed on the GUI.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>defaultProperty</td>
<td>The workstep name.</td>
</tr>
<tr>
<td>stringValue</td>
<td>The workstep name.</td>
</tr>
<tr>
<td>workstepName</td>
<td>The workstep name.</td>
</tr>
<tr>
<td>escapedWorkstepName</td>
<td>The workstep name modified in order to avoid using spaces and $ signs. The escape character is <em>.</em></td>
</tr>
<tr>
<td>processTemplateName</td>
<td>The process template name.</td>
</tr>
</tbody>
</table>

Arguments

The **processIndex** attribute of the `<argument>` tag contains the index of the ProcessArgs object in the arguments array. The **workstepIndex** attribute of the `<argument>` tag contains the index of the workstep in the `selectedWorksteps` array.

**Macro type: dataslot**

Allows the user to choose a dataslot from a combo box.

**Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>defaultProperty</td>
<td>The dataslot name.</td>
</tr>
<tr>
<td>stringValue</td>
<td>The dataslot name.</td>
</tr>
<tr>
<td>dataslotName</td>
<td>The dataslot name.</td>
</tr>
<tr>
<td>dataslotType</td>
<td>The dataslot type.</td>
</tr>
</tbody>
</table>
Input properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>dataslotName</td>
<td>The dataslot name.</td>
</tr>
<tr>
<td>dataslotFilter</td>
<td>The valid value is numeric. If specified, then only numeric dataslots are presented to the user. If not present, then the combo box contains all the dataslots accessible from the rule language.</td>
</tr>
</tbody>
</table>

**Macro type: completeworkstep-action**

Specifies the data required for completing a workstep.

**Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>defaultProperty</td>
<td>The rule language statements required to complete the work item and assign values to some dataslots.</td>
</tr>
<tr>
<td>stringValue</td>
<td>The rule language statements required to complete the work item and assign values to some dataslots.</td>
</tr>
</tbody>
</table>

Input properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>dataslots-prop</td>
<td>A hash property containing dataslot names as keys and values assigned to these dataslots as strings.</td>
</tr>
</tbody>
</table>

**Macro type: sendmail-action**

Specifies the data necessary for sending an e-mail.

**Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>defaultProperty</td>
<td>The rule language statements required to send an e-mail with the specified fields.</td>
</tr>
<tr>
<td>stringValue</td>
<td>The rule language statements required to send an e-mail with the specified fields.</td>
</tr>
</tbody>
</table>
Input properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>to</td>
<td>A string for the <strong>To</strong> field of the e-mail.</td>
</tr>
<tr>
<td>from</td>
<td>A string for the <strong>From</strong> field of the e-mail.</td>
</tr>
<tr>
<td>subject</td>
<td>A string for the <strong>Subject</strong> field of the e-mail.</td>
</tr>
<tr>
<td>body</td>
<td>A string for the body of the e-mail.</td>
</tr>
</tbody>
</table>

In cases where the input properties contain an expression, such as `<dataslot dataslotname>` and `<performer performername>`, a substitution occurs when the e-mail message is sent with the actual value of the dataslot or the performer (a dataslot or a plain user name).

**Macro type: reassign-action**

Specifies the data required for reassigning a work item.

**Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>defaultProperty</td>
<td>The rule language statements required to reassign the work item.</td>
</tr>
<tr>
<td>stringValue</td>
<td>The rule language statements required to reassign the work item.</td>
</tr>
</tbody>
</table>

**Input properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>reassignTo</td>
<td>A string containing the new performer of the work item. May contain an expression such as <code>&lt;dataslot dataslotname&gt;</code> indicating replacement of the actual value of the dataslot.</td>
</tr>
</tbody>
</table>

**Macro type: changepriority-action**

Specifies the data necessary for changing work item priority.
Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>defaultProperty</td>
<td>The rule language statements required to change the priority of the work item.</td>
</tr>
<tr>
<td>stringValue</td>
<td>The rule language statements required to change the priority of the work item.</td>
</tr>
</tbody>
</table>

Input properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>selectedRadio</td>
<td>The valid values:</td>
</tr>
<tr>
<td></td>
<td>• radPriority. To change the priority to the level indicated by the priority input property.</td>
</tr>
<tr>
<td></td>
<td>• radUp. To change the priority one level up.</td>
</tr>
<tr>
<td></td>
<td>• radDown. To change the priority one level down.</td>
</tr>
<tr>
<td>priority</td>
<td>Valid values are low, medium, and high. If the selectedRadio property contains the &quot;&quot; value, then the priority changes to the value of this property.</td>
</tr>
</tbody>
</table>

References

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>workstepName</td>
<td>An expression of type macro.property which gives the workstep name the rule template is applied to. The value must be non-null.</td>
</tr>
</tbody>
</table>

Macro type: module

To allow the user to choose the file where the generated rules are saved. The module name is chosen implicitly (as it must be identical to the file name).

Default file name and path rules

The macro proposes a default file name and a path using the following rules:

• If the rule template applies to a specific workstep, the file name is:
  `<rule_template_name>_<workstep_name>.txt`

• If the rule template applies to the entire process template, then the file name is:
  `<rule_template_name>.txt`

• By default, the file is saved in the Business Process Server application's rules directory, if this directory exists, otherwise the user can select where to save.
How to write additional macros and editors

You may add custom macros and editors to the Rule Wizard Framework by writing specific Java classes. Custom macro classes must implement the MacroDataIF interface. DefaultMacroData is an abstract class which implements some of the members of MacroDataIF is provided in order to speed up development. In addition, custom editor classes implement the EditorIF interface and derive from the JPanel swing class. Use DefaultEditor, an abstract class which derives from JPanel and implements some EditorIF members, as the base class for custom editors.

In order to use a custom macro, specify the fully qualified class name in the class attribute of the <macro> tag instead of using the type attribute.

Usually the Java class implementing an editor is inferred by examining the type of edited macro. The <editor> tag also supports a class attribute intended for use when you want to explicitly specify the Java class.

The next two sections provide sample macro and editor implementations for the boolean type.
Sample macro implementation

The following is a sample implementation of a simple macro, the `boolean` type. The source code containing the verbose comments is provided. The next section provides associated editor implementation for this type, using a checkbox GUI component.

```java
package com.savvion.rulewizard.data;
import java.util.Hashtable;
/**
 * Macro class implementing the data structure used by the "boolean" type.
 */
public class BooleanMacro extends DefaultMacroData {
    protected boolean boolValue;
    public static final String STRING_VALUE = "stringValue";

    /**
     * Get the boolean value of this macro.
     * This property will be used from Velocity programs and CheckBoxEditor.
     * @return value of boolValue.
     */
    public boolean getBoolValue() {
        return boolValue;
    }

    /**
     * Set the boolean value of this macro.
     * This property will be used from Velocity programs and CheckBoxEditor. @param v Value to assign to boolValue.
     */
    public void setBoolValue(boolean v) {
        this.boolValue = v;
    }

    /**
     * @return a string representation of this macro's value ("true"
     * or "false" string)
     */
    public String getStringValue() {
        return (new Boolean(boolValue)).toString();
    }

    /**
     * Initializes this macro with the value represented by the string in
     * the parameter. * @param v a string representation of this macro's value ("true"
     * or "false" string)
     */
    public void setStringValue(String v) {
        this.boolValue = Boolean.valueOf(v).booleanValue();
    }

    /**
     * Although no validation is required we implement this method because
     * the base class (DefaultMacroData) does not. * @return always the boolean "true" because the data in this macro is
     * always valid
     */
    public boolean verify() {
        return true;
    }

    /**
     * Called by the persistence manager which saves the parameter values
     * once a rule template has been customized. * We have to save the value of this macro as a string so we use the
     * getStringValue() method. Based on the value returned by this method
     * the persistence manager will create a tag like:
     */
```
Sample editor implementation

This section presents the Java class implementing the editor for the boolean type. All implementation issues are commented in the source file.

```java
package com.savvion.rulewizard.gui;
import com.savvion.rulewizard.data.MacroDataIF;
import com.savvion.rulewizard.data.BooleanMacro;
import java.util.Hashtable;
import javax.swing.JCheckBox;
import java.awt.BorderLayout;

public class CheckBoxEditor extends DefaultEditor {
    JCheckBox field = new JCheckBox();
    BooleanMacro boolMacro; // macro holding the data

    public CheckBoxEditor() {
        super();
        add(field, BorderLayout.WEST);

        if (boolMacro != null) {
            field.setSelected(boolMacro.getValue());
        }

        field.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                if (field.isSelected()) {
                    boolMacro.setValue(true);
                } else {
                    boolMacro.setValue(false);
                }
            }
        });
    }

    @Override
    public void setMacro(MacroDataIF macro) {
        super.setMacro(macro);
        boolMacro = (BooleanMacro) macro; // macro holding the data
    }
}
```
Transfer the data from GUI to the macro.

```java
public void save() {
    boolMacro.SetBoolValue(field.isSelected());
}
```

Transfer the data from macro to the GUI.

```java
public void load() {
    field.setSelected(boolMacro.getBoolValue());
}
```

There are no validations to perform. We always return "true".

```java
public boolean verify() {
    return true;
}
```

Get the associated macro.

```java
public MacroDataIF getMacroData() {
    return (MacroDataIF)boolMacro;
}
```

Associate this editor with a macro. We cast the macro object to BooleanMacro as we know this is the only kind of macro this editor works with.

```java
public void setMacroData(MacroDataIF v) {
    this.boolMacro = (BooleanMacro)v;
}
```

If any data-dependent GUI initialization is needed this is the place to do it because showGUI is called after setMacroData(). For this editor there is none.

```java
public void showGUI() {}
```

Called when this editor is to receive focus. We put the focus on the check box.

```java
public void setFocus() {
    field.requestFocus();
}
```

Enable or disable the editor according to the value of parameter b.

```java
public void setEnabled(boolean b) {
    field.setEnabled(b);
}
```
Skipping a macro

The following template provides a simple technique that enables you to decide at run time what macros to show in one of the wizard steps.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE template SYSTEM "resources/ruleTemplate.dtd" >
<template name="test">
  <label>
    <istring lang="en">test</istring>
  </label>
  <screen number="10">
    <editor macro="list" label="List"/>
  </screen>

  <screen number="20">
    <editor macro="ref1" label="Reference 1"
      class="com.savvion.rulewizard.gui.TextEditor"/>
  </screen>

  <screen number="25">
    <editor macro="ref2" label="Reference 2"
      class="com.savvion.rulewizard.gui.TextEditor"/>
  </screen>

  <screen number="30">
    <editor macro="text" label="Text"/>
  </screen>

  <macro name="list" type="combo-box">
    <vector-property name="listLabels">
      <vector-elem>Optional macro 1</vector-elem>
      <vector-elem>Optional macro 2</vector-elem>
      <vector-elem>None</vector-elem>
    </vector-property>
    <vector-property name="listValues">
      <vector-elem>ref1</vector-elem>
      <vector-elem>ref2</vector-elem>
      <vector-elem>none</vector-elem>
    </vector-property>
    <property name="listSelectedValue">option_a</property>
  </macro>

  <macro name="ref1" class="MyVarname">
    <reference name="option" value="list.selectedValue"/>
    <property name="show_for_option">ref1</property>
  </macro>

  <macro name="ref2" class="MyVarname">
    <reference name="option" value="list.selectedValue"/>
    <property name="show_for_option">ref2</property>
  </macro>

  <macro name="text" type="varname">
  </macro>

  <text>
    Value of list is $list.selectedValue.
    Value of ref1 is $ref1.
    Value of ref2 is $ref2.
    Value of text is $text.
  </text>
</template>
```
When you run the above template, what you see in the second step of the Wizard depends on the value chosen in the list box presented in the first step. For example, if the chosen value is "Optional macro 1," then you will receive the macro ref1 in the second step. Otherwise, you will receive the macro ref2. Although the same editor is used for both macros, note the difference in the labels, "Reference 1" and "Reference 2", as well as the changes in the Wizard output.

The following example shows the Wizard output in the case where "Optional macro 2" is chosen.

**Figure 29: Optional Macro 2 choice**

![Wizard Output Example](image)

The output for this example is as follows:

- Value of list is ref2.
- Value of ref1 is null.
- Value of ref2 is value2.
- Value of text is text_value.

In the template, the screen numbered 20 was skipped because the only macro it contained, ref1, returned the value SKIP from the method getSkipStatus. The following code of the MyVarname Java class, represents the macro type for the macros ref1 and ref2.

```java
import java.util.Hashtable;
import com.savvion.rulewizard.data.*;
import com.savvion.rulewizard.util.Debugger;
public class MyVarname extends MacroString {
    String show_for_option = null;
    public int getSkipStatus() {
        if (show_for_option == null)
            return NO_SKIP;
        Debugger.println("MyVarname.getSkipStatus");
        String option = (String)references.get("option");
        ...
    }
}
```
if (option != null) {
    String optionName = (String)macroContext.getReference(option);
    Debugger.println("MyVarname: option " + optionName);
    return show_for_option.equalsIgnoreCase(optionName) ? NO_SKIP :
               SKIP;
} else
    return NO_SKIP;

public void addProperty(String name, String value) {
    super.addProperty(name, value);
    if (name != null && name.equals("show_for_option"))
        show_for_option = value;
}

public Hashtable getPropertiesToSave() {
    Hashtable pts = super.getPropertiesToSave();
    if (show_for_option != null)
        pts.put("show_for_option", show_for_option);
    return pts;
}

The addProperty and getPropertiesToSave methods just read and save the property show_for_option, which indicates which value of the option reference the macro should be present. The getSkipStatus method reads the value of the option reference and compares it to the value of the property show_for_option. If the values are equal, then the NO_SKIP value returns, meaning the macro is present. Otherwise, SKIP is returned, meaning the macro is invisible.
BPM Process Store is an open repository of rich business information collected from different modules of Business Process Server. The BPM Process Store server processes events and populates Business Process Server system tables. This chapter describes the BPM Process Store server, the Business Process Server database schema, and the Business Process Server event management model.

This chapter also describes the Dashboard database tables on page 170, which provide statistical information that is displayed in widgets in the Management module of Business Process Portal.

For details, see the following topics:

• Understanding the BPM Process Store database
• Business Process Server event management
• Schema for process tables
• Dashboard database tables
Understanding the BPM Process Store database

As a repository for business intelligence, the BPM Process Store database maintains two types of information stored as two sets of tables—Event tables and Process tables. Business Process Server uses these tables as part of its event driven architecture to manage the information flow. The Event tables include BizEvent, BPM Process StoreEventCounter, and BPM EventsEventCounter. For example, BP Server can generate events and they are stored in the BPM Events table (for more information, see Schema for event tables on page 155). The Process tables consist of the process template, ProcessXML, process instance, Workstep, work item, and additional tables for metadata information (for more information, see Schema for process tables on page 164).

Note: In earlier Business Process Server releases, BP Server never actually read any data from the database. Instead, there were two consumers for BP Server events: BPM Process Store and BPM Events. Now Business Process Server uses separate BP Server database tables to maintain process transactional data for recovery of active process instance and applications.

BPM Process Store reads events from the Event tables and determines the process state change information encapsulated by the events. BPM Process Store then uses this information to populate the Process tables, including the process instance, Workstep, and work item tables in addition to meta data tables. These tables store information such as the current state of process instances, start date, and end date as well as the current state of worksteps and workitems.

BPM Events is the other consumer of BP Server information. BPM Events reads events from the Event tables and evaluates its rules against these events. BPM Events populates the infopads, then stores these infopads into database tables.

The Management module of Business Process Portal is the ultimate consumer of information stored in the Process tables. It uses this information to prepare management reports that summarize the business intelligence and to monitor and control processes.

The information accessed by the Management module of Business Process Portal reflects a historical snapshot of a business state which may not be the most current state of process instances and infopads. The BP Server maintains the most current state of the process instances as Java objects. For the Management module to obtain the most current process instances, it must call BP Server API directly. Similarly, it can obtain the most current infopads by calling the BPM Events API directly.

Note: Tables containing event and process instance information may occupy large database space. If no action is taken, then all storage currently allocated to the database is consumed eventually, and Business Process Server will not run properly. We recommend you unload events, process instances, and infopads by archiving them periodically with the "autodelete" option after those events, process instances, and infopads are no longer required in the database. Refer to Server Administrator’s Guide for detailed information on archiving. You may also choose to increase the database size (table space size) as well.

Business Process Server event management

Events are notifications that the BP Server sends out to inform BPM Process Store, BPM Events, and other components about the process and data changes that have occurred in BP Server. These components can retrieve and process the events stored in the BPM Events database table. This section describes how events are represented at run time, and how events are represented in the database.
Event and CommonEvent class

An event record contains the following information: event_id, type, value, date, and context_object. Type is a string that represents the origin of the event. An example of the type field is "BPServer". Event_id is the ID of the event. Process_instance_id is the ID of the process instance on which the event is sent. Value is a string that further specifies the semantics of each event. Examples of value include "PI_ACTIVATED" and "W_COMPLETED" which means "process instance activated" and "workstep completed" respectively.

Business Process Server events

The list of all events and their corresponding descriptions are presented in the following table:

Table 15: Business Process Server events

<table>
<thead>
<tr>
<th>Event value</th>
<th>Event description</th>
<th>ID number</th>
</tr>
</thead>
<tbody>
<tr>
<td>P_CREATED</td>
<td>A process template is created</td>
<td>1</td>
</tr>
<tr>
<td>P_INSTALLED</td>
<td>A process template is installed</td>
<td>2</td>
</tr>
<tr>
<td>P_REMOVED</td>
<td>A process template is removed</td>
<td>3</td>
</tr>
<tr>
<td>P_RESUMED</td>
<td>A process template is resumed</td>
<td>5</td>
</tr>
<tr>
<td>P_SUSPENDED</td>
<td>A process template is suspended</td>
<td>4</td>
</tr>
<tr>
<td>PI_CREATED</td>
<td>A process instance is created</td>
<td>6</td>
</tr>
<tr>
<td>PI_ACTIVATED</td>
<td>A process instance is activated</td>
<td>7</td>
</tr>
<tr>
<td>PI_SUSPENDED</td>
<td>A process instance is suspended</td>
<td>8</td>
</tr>
<tr>
<td>PI_RESUMED</td>
<td>A process instance is resumed</td>
<td>9</td>
</tr>
<tr>
<td>PI_COMPLETED</td>
<td>A process instance is completed</td>
<td>11</td>
</tr>
<tr>
<td>PI_PRIORITYSET</td>
<td>The priority of a process instance is changed</td>
<td>13</td>
</tr>
<tr>
<td>PI_DUEDATESET</td>
<td>The due date of a process instance is changed</td>
<td>14</td>
</tr>
<tr>
<td>PI_REMOVED</td>
<td>A process instance is removed</td>
<td>12</td>
</tr>
<tr>
<td>PI_CREATORSET</td>
<td>A process instance creator is set</td>
<td>51</td>
</tr>
<tr>
<td>PI_ATTRIBUTESET</td>
<td>Process instance attributes are set in bulk</td>
<td>41</td>
</tr>
<tr>
<td>PI_DATASLOTSSET</td>
<td>Process instance dataslots are set in bulk</td>
<td>42</td>
</tr>
<tr>
<td>W_CREATED</td>
<td>A workstep is created</td>
<td>15</td>
</tr>
<tr>
<td>W_ACTIVATED</td>
<td>A workstep is activated</td>
<td>18</td>
</tr>
<tr>
<td>W_SUSPENDED</td>
<td>A workstep is suspended</td>
<td>16</td>
</tr>
<tr>
<td>Event value</td>
<td>Event description</td>
<td>ID number</td>
</tr>
<tr>
<td>--------------------------</td>
<td>---------------------------------------------------------------</td>
<td>-----------</td>
</tr>
<tr>
<td>W_RESUMED</td>
<td>A workstep is resumed</td>
<td>17</td>
</tr>
<tr>
<td>W_COMPLETED</td>
<td>A workstep is completed</td>
<td>21</td>
</tr>
<tr>
<td>W_PRIORITYSET</td>
<td>The priority of a workstep is changed</td>
<td>22</td>
</tr>
<tr>
<td>W_DUEDATESET</td>
<td>The due date of a workstep is changed</td>
<td>23</td>
</tr>
<tr>
<td>W_PERFORMERSET</td>
<td>The performer of a workstep is changed</td>
<td>24</td>
</tr>
<tr>
<td>W_EVENTACTIVATION_WAIT</td>
<td>A monitoring workstep enters into wait state for an activation event.</td>
<td>91</td>
</tr>
<tr>
<td>I_CREATED</td>
<td>A workitem is created</td>
<td>25</td>
</tr>
<tr>
<td>I_ACTIVATED</td>
<td>A work item is activated</td>
<td>26</td>
</tr>
<tr>
<td>I_AVAILABLE</td>
<td>A work item is available</td>
<td>27</td>
</tr>
<tr>
<td>I_ASSIGNED</td>
<td>A work item is assigned</td>
<td>28</td>
</tr>
<tr>
<td>I_COMPLETED</td>
<td>A work item is completed</td>
<td>31</td>
</tr>
<tr>
<td>I_PRIORITY</td>
<td>A work item priority is changed</td>
<td>47</td>
</tr>
<tr>
<td>I_DUEDATE</td>
<td>A work item due date is changed</td>
<td>46</td>
</tr>
<tr>
<td>I_PERFORMER</td>
<td>A work item performer is changed</td>
<td>45</td>
</tr>
<tr>
<td>I_SUSPENDED</td>
<td>A work item is suspended</td>
<td>32</td>
</tr>
<tr>
<td>S_UPDATED</td>
<td>A dataslot is updated</td>
<td>33</td>
</tr>
<tr>
<td>EP_AFTERBREAK</td>
<td>Able to restart and resume the broken adapter</td>
<td>35</td>
</tr>
</tbody>
</table>

The event occurring time is stored in the date attribute as a long integer. The real event data is stored in the context attribute. The context is represented as a Map. A composite object can be represented using a Map from Java.

**Event object example**

An example of an event object is shown below.

```java
EVENT_ID: 582
TYPE: BP Server
EVENT_TYPE: W_ACTIVATED
CREATE_DATE: Fri Sep 28 16:29:50 IST 2012 (This is converted from timestamp of long type)
PROCESS_TEMPLATE_ID: 40
PROCESS_INSTANCE_ID: 584
WORKSTEP_ID: 4
EVENT_CONTEXT_OBJ:
    {APP_NAME=TestApp,
     PROCESSTEMPLATENAME=TestApp,
     PROCESSTEMPLATEID=40,
```
Process instance:

```java
PROCESSINSTANCEID=584,
PROCESSINSTANCENAME=TestApp#584,
WORKSTEPNAME=Activity 3, WORKSTEPID=4,
TYPE=ATOMIC
MONITOR_STEP=false,
SBM_VERSION=9.0.0 (build #205),
STARTTIME=1348829989923,
PERFORMERVALUE=ebms,
DUEDATE=1348837189923,
PRIORITY=medium,
LOOPCOUNTER=1,
SESSION_USER=ebms,
RPID=0,
PERFORMER=@CREATOR,
ENDTIME=0,
ESTIMATEDDURATION=7200,
}
```

Schema for event tables

From the perspective of a BPM Process Store developer, the most important aspect of event management is the definition of the Event tables—BPM Process Store EventCounter, and BPM Events EventCounter. Business Process Server uses these tables for storing events, and for storing BPM Process Store and BPM Events event counter values, respectively.

**BizEvent table**

In the BizEvent table, EVENT_ID uniquely identifies an event and is generated by the database server in order to guarantee its uniqueness. The result set retrieved from BizEvent should be sorted on EVENT_ID because the event processing in BPM Process Store assumes the order of the event as the order of the event creation. The columns TYPE, VALUE, and CREATE_DATE store the corresponding values in a CommonEvent object. The column CONTEXT_OBJ stores event context in a serialized Java object format.

**BizEvent table fields**

The definition of the BizEvent table fields are as follows:

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EVENT_ID</td>
<td>NUMBER NOT NULL</td>
<td>Unique event identifier</td>
</tr>
<tr>
<td>TYPE</td>
<td>VARCHAR2(30) NOT NULL</td>
<td>Event type</td>
</tr>
<tr>
<td>VALUE</td>
<td>VARCHAR2(20) NOT NULL</td>
<td>Value of the event</td>
</tr>
<tr>
<td>CREATE_DATE</td>
<td>NUMBER NOT NULL</td>
<td>Event creation time</td>
</tr>
<tr>
<td>CONTEXT_OBJ</td>
<td>LONG RAW</td>
<td>Stores event in Java serialized object format.</td>
</tr>
<tr>
<td>PROCESS_INSTANCE_ID</td>
<td>NUMBER</td>
<td>The process instance ID</td>
</tr>
</tbody>
</table>

3 Table primary key
BPM Process Store Event Counter table

The BPM Process Store Event Counter table stores the last event ID processed by BPM Process Store. In the case of system failure, BPM Process Store issues a query against BPM Process Store Event Counter to determine the last processed event ID in the previous execution. Using this event ID, the BPM Process Store can then process all unprocessed events in the event table to bring the process tables into a consistent state identical to the one just before the failure. Business Process Server can then continue from this state without loss of business data.

BPM Process Store Event Counter table fields

The definition of the BPM Process Store Event Counter table fields are as follows:

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LAST_EVENT_ID</td>
<td>NUMBER NOT NULL</td>
<td>The last event processed by BPM Process Store.</td>
</tr>
<tr>
<td>LAST_UPDATE_DATE</td>
<td>DATE</td>
<td>The time of the last event processed by BPM Process Store.</td>
</tr>
</tbody>
</table>

BPM Events Event Counter table

Similarly, the BPM Events Event Counter table stores the last event ID processed by BPM Events. During failure recovery, BPM Events uses this ID to query all unprocessed events and then processes them. At this point, the infopad maintained by BPM Events is synchronized with the rest of the system.

BPM Events Event Counter table fields

The definition of the BPM Events Event Counter table fields are as follows:

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LAST_EVENT_ID</td>
<td>NUMBER NOT NULL</td>
<td>The last event processed by BPM Events.</td>
</tr>
<tr>
<td>LAST_UPDATE_DATE</td>
<td>DATE</td>
<td>The time of the last event processed by BPM Events.</td>
</tr>
</tbody>
</table>

Note: The hidden parameter, `bpmprocessstore.events.transaction.size` is applicable only in the case of an asynchronous mode (stand-alone classic mode). If not available in `bpserver.conf`, then the value of this parameter is 10. This means 10 events are processed in a single transaction to improve efficiency. You can set this parameter to the value of your choice.
Event database keys

The following sections describe the keys and event values for ProcessEvents, ProcessInstance, Workstep Events, Work Item Events, and Dataslot Events.

ProcessEvents' keys and event values

The following table lists the keys and event values for Process Events.

<table>
<thead>
<tr>
<th>Keys</th>
<th>Event values</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>P_CREATED</td>
</tr>
<tr>
<td>PROCESSTEMPLATEID</td>
<td>M⁴</td>
</tr>
<tr>
<td>DURATION</td>
<td>M</td>
</tr>
<tr>
<td>PMANAGER</td>
<td>M</td>
</tr>
<tr>
<td>PROCESSTEMPLATENAME</td>
<td>M</td>
</tr>
<tr>
<td>PROCESSWSLIST</td>
<td>M</td>
</tr>
<tr>
<td>PROCESSDATA</td>
<td>M</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>M</td>
</tr>
<tr>
<td>CATEGORY</td>
<td>Optional</td>
</tr>
<tr>
<td>GROUP</td>
<td>Optional</td>
</tr>
</tbody>
</table>

ProcessInstance's events keys and event values

The following table lists the keys and event values for ProcessInstance Events.

---

⁴ M: Mandatory
### Keys and event values

<table>
<thead>
<tr>
<th>Keys</th>
<th>Event</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>PI_</td>
<td>CREATORSET</td>
<td></td>
</tr>
<tr>
<td>PI_</td>
<td>DUEDATESET</td>
<td></td>
</tr>
<tr>
<td>PI_</td>
<td>PRIORITYSET</td>
<td></td>
</tr>
<tr>
<td>PI_</td>
<td>REMOVED</td>
<td></td>
</tr>
<tr>
<td>PI_</td>
<td>COMPLETED</td>
<td></td>
</tr>
<tr>
<td>PI_</td>
<td>RESUMED</td>
<td></td>
</tr>
<tr>
<td>PI_</td>
<td>SUSPENDED</td>
<td></td>
</tr>
<tr>
<td>PI_</td>
<td>ACTIVATED</td>
<td></td>
</tr>
<tr>
<td>PI_</td>
<td>CREATED</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Keys</th>
<th>Template ID</th>
<th>Duration</th>
<th>RPID</th>
<th>Template Name</th>
<th>Start Time</th>
<th>End Time</th>
<th>Creator</th>
<th>Process ID</th>
<th>Process Name</th>
<th>Priority</th>
<th>Parent ID</th>
<th>Caller ID</th>
<th>Caller Name</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

OpenEdge Business Process Server: Application Developer's Guide

Chapter 11: Understanding the Business Process Server repository
**Workstep Events' keys and event values**

The following table lists the keys and event values for Workstep Events.

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

5 M: Mandatory
7 Sub: Applicable only to subprocess instance
6 N/A: Not Applicable
8 Opt: Optional
## Keys and event values

<table>
<thead>
<tr>
<th>Key</th>
<th>Event</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>EP_</td>
<td>AFTERBREAK</td>
<td></td>
</tr>
<tr>
<td>EP_</td>
<td>BROKEN</td>
<td></td>
</tr>
<tr>
<td>W_</td>
<td>PERFORMERSET</td>
<td></td>
</tr>
<tr>
<td>W_</td>
<td>DUEDATESET</td>
<td></td>
</tr>
<tr>
<td>W_</td>
<td>PRIORITYSET</td>
<td></td>
</tr>
<tr>
<td>W_</td>
<td>RESUMED</td>
<td></td>
</tr>
<tr>
<td>W_</td>
<td>ACTIVATED</td>
<td></td>
</tr>
<tr>
<td>W_</td>
<td>COMPLETED</td>
<td></td>
</tr>
<tr>
<td>W_</td>
<td>SUSPENDED</td>
<td></td>
</tr>
<tr>
<td>MMMM</td>
<td>PROCESSTEMPLATEID</td>
<td></td>
</tr>
<tr>
<td>MMMM</td>
<td>DURATION</td>
<td></td>
</tr>
<tr>
<td>MMMM</td>
<td>RPID</td>
<td></td>
</tr>
<tr>
<td>MMMM</td>
<td>PROCESSTEMPLATENAME</td>
<td></td>
</tr>
<tr>
<td>MMMM</td>
<td>STARTTIME</td>
<td></td>
</tr>
<tr>
<td>MMMM</td>
<td>ENDTIME</td>
<td></td>
</tr>
<tr>
<td>MMMM</td>
<td>LOOPCOUNTER</td>
<td></td>
</tr>
<tr>
<td>MMMM</td>
<td>PROCESSINSTANCEID</td>
<td></td>
</tr>
<tr>
<td>MMMM</td>
<td>PROCESSINSTANCENAME</td>
<td></td>
</tr>
<tr>
<td>MMMM</td>
<td>PRIORITY</td>
<td></td>
</tr>
<tr>
<td>MMMM</td>
<td>PERFORMER</td>
<td></td>
</tr>
<tr>
<td>MMMM</td>
<td>WORKSTEPNAME</td>
<td></td>
</tr>
<tr>
<td>MMMM</td>
<td>WORKSTEPID</td>
<td></td>
</tr>
</tbody>
</table>

9 Mandatory
<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event</td>
<td>N/A</td>
<td>Values</td>
<td>N/A</td>
</tr>
<tr>
<td>dataslots</td>
<td>10</td>
<td>Keys</td>
<td>N/A</td>
</tr>
<tr>
<td>INSTRUCTION</td>
<td>M M M M M M M M M</td>
<td>DUE_DATE</td>
<td>N/A</td>
</tr>
<tr>
<td>PIPR_dsname</td>
<td>(the dataslot mapped to the process instance priority)</td>
<td>WSPR_dsname</td>
<td>(the dataslot mapped to the workstep due date)</td>
</tr>
<tr>
<td>WSDD_dsname</td>
<td>(the dataslot mapped to the workstep priority)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Work Item Events' keys and event values

The following table lists the keys and event values for Work Item Events.

<table>
<thead>
<tr>
<th>Event</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>I_PRIORITY</td>
<td></td>
</tr>
<tr>
<td>I_DUEDATE</td>
<td></td>
</tr>
<tr>
<td>I_SUSPENDED</td>
<td></td>
</tr>
<tr>
<td>I_REMOVED</td>
<td></td>
</tr>
<tr>
<td>I_PERFORMER</td>
<td></td>
</tr>
<tr>
<td>I_COMPLETED</td>
<td></td>
</tr>
<tr>
<td>I_ASSIGNED</td>
<td></td>
</tr>
<tr>
<td>IAVAILABLE</td>
<td></td>
</tr>
<tr>
<td>ICREATED</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Keys</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>MM</td>
<td>M M M M M M M M</td>
</tr>
<tr>
<td>MM</td>
<td>M M M M M M M M</td>
</tr>
<tr>
<td>MM</td>
<td>M M M M M M M M</td>
</tr>
<tr>
<td>MM</td>
<td>M M M M M M M M</td>
</tr>
<tr>
<td>MM</td>
<td>M M M M M M M M</td>
</tr>
<tr>
<td>MM</td>
<td>M M M M M M M M</td>
</tr>
<tr>
<td>MM</td>
<td>M M M M M M M M</td>
</tr>
<tr>
<td>MM</td>
<td>M M M M M M M M</td>
</tr>
<tr>
<td>MM</td>
<td>M M M M M M M M</td>
</tr>
<tr>
<td>MM</td>
<td>M M M M M M M M</td>
</tr>
<tr>
<td>MM</td>
<td>M M M M M M M M</td>
</tr>
<tr>
<td>MM</td>
<td>M M M M M M M M</td>
</tr>
<tr>
<td>MM</td>
<td>M M M M M M M M</td>
</tr>
<tr>
<td>MM</td>
<td>M M M M M M M M</td>
</tr>
<tr>
<td>MM</td>
<td>M M M M M M M M</td>
</tr>
<tr>
<td>MM</td>
<td>M M M M M M M M</td>
</tr>
<tr>
<td>MM</td>
<td>M M M M M M M M</td>
</tr>
<tr>
<td>MM</td>
<td>M M M M M M M M</td>
</tr>
<tr>
<td>MM</td>
<td>M M M M M M M M</td>
</tr>
<tr>
<td>MM</td>
<td>M M M M M M M M</td>
</tr>
<tr>
<td>MM</td>
<td>M M M M M M M M</td>
</tr>
<tr>
<td>MM</td>
<td>M M M M M M M M</td>
</tr>
<tr>
<td>MM</td>
<td>M M M M M M M M</td>
</tr>
<tr>
<td>MM</td>
<td>M M M M M M M M</td>
</tr>
<tr>
<td>MM</td>
<td>M M M M M M M M</td>
</tr>
</tbody>
</table>

10 N/A: Not Applicable
11 Opt: Optional
Dataslot Events' keys and event values

The following table lists the keys and event values for Dataslot Events.

Keys and event values

<table>
<thead>
<tr>
<th>Keys</th>
<th>Event values</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>S_UPDATED</td>
</tr>
<tr>
<td>PROCESSTYPE</td>
<td>M^13</td>
</tr>
<tr>
<td>IS_GLOBAL</td>
<td>M</td>
</tr>
<tr>
<td>RPID</td>
<td>M</td>
</tr>
<tr>
<td>PROCESSTYPE</td>
<td>M</td>
</tr>
<tr>
<td>DATASLOTNAME</td>
<td>M</td>
</tr>
<tr>
<td>DATASLOTVALUE</td>
<td>M</td>
</tr>
<tr>
<td>DATASLOTTYPE</td>
<td>M</td>
</tr>
<tr>
<td>PROCESSINSTANCEID</td>
<td>M</td>
</tr>
<tr>
<td>PROCESSINSTANCE</td>
<td>M</td>
</tr>
</tbody>
</table>

^12 M: Mandatory
^13 M: Mandatory
Schema for process tables

This section describes the schema for Process tables. Business Process Server stores process templates, process instances, worksteps, workitems, and metadata information in a set of relational database tables.

Business Process Server uses a sequence generator to generate IDs for PROCESTEMPLETABLE, PROCESSINSTANCE, and WORKITEM. For every sequence (for example, processinstance sequence), the BP Server stores 100 numbers in memory cache in order to avoid excessive database access. If the user stops the BP Server using CTRL+C, then all the cached numbers are lost, causing a gap among the IDs.

Process template tables

Business Process Server uses two tables, PROCESTEMPLETABLE and PROCESSXML, to store process templates.

PROCESTEMPLETABLE table

The PROCESTEMPLETABLE table stores information about process templates, including its ID, name, manager, and table name for dataslots. The following table presents the definition of the PROCESTEMPLETABLE table fields.

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROCESS_TEMPLATE_ID</td>
<td>NUMBER NOT NULL</td>
<td>Unique template ID for a process</td>
</tr>
<tr>
<td>PROCESS_TEMPLATE_NAME</td>
<td>VARCHAR2(64) NOT NULL</td>
<td>Unique process template name</td>
</tr>
<tr>
<td>PROCESS_MANAGER</td>
<td>VARCHAR2(32)</td>
<td>Manager of the process</td>
</tr>
<tr>
<td>INSTANCE_DATA_TABLE_NAME</td>
<td>VARCHAR2(30)</td>
<td>None</td>
</tr>
<tr>
<td>GLOBAL_DATA_TABLE_NAME</td>
<td>VARCHAR2(30)</td>
<td>None</td>
</tr>
<tr>
<td>PROCESS_INSTANCE_COUNTER</td>
<td>NUMBER</td>
<td>The process instance counter</td>
</tr>
<tr>
<td>STATUS</td>
<td>CHAR(20) NOT NULL</td>
<td>Process template state such as P_CREATED, P_INSTALLED, or P_SUSPENDED.</td>
</tr>
<tr>
<td>APP_NAME</td>
<td>VARCHAR2(64)</td>
<td>Application name that is common across all versions of an application</td>
</tr>
<tr>
<td>PROCESS_TYPE</td>
<td>NUMBER(1)</td>
<td>Type of process – business process, system process</td>
</tr>
<tr>
<td>MONITOR_DS_NAME</td>
<td>VARCHAR2(32)</td>
<td>The dataslot name of the external instance ID for a process containing a monitoring workstep</td>
</tr>
</tbody>
</table>

Table constraint: PROCESS_TEMPLATE_PKEY
PROCESSXML table
The PROCESSXML table\(^\text{15}\) stores XML files for all created Business Process Server application templates. The following table presents the definition of the PROCESSXML table fields.

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROCESS_TEMPLATE_ID</td>
<td>NUMBER NOT NULL</td>
<td>Unique template ID for a process</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>VARCHAR2(1000)</td>
<td>Process/application description</td>
</tr>
<tr>
<td>XML</td>
<td>LONG</td>
<td>Process template in XML format</td>
</tr>
</tbody>
</table>

Dataslot tables for a process template
BPM Process Store creates the following two run-time tables for each process template installed:

- An instance level dataslot table, which uses the process template name as the name of the table. It consists of all the instance level dataslot values. It also contains two key fields, PROCESS_TEMPLATE_ID and PROCESS_INSTANCE_ID, used to uniquely identify each row. Every process instance of the same process template has a corresponding row in this table.

- A global dataslot table, which uses the PROCESSTEMPLATEID in the name of the table as BS_<ptid>. It consists of all global dataslots for the corresponding PROCESSTEMPLATE. It also contains PROCESS_TEMPLATE_ID as one of the column. Generally, there is only one row in this table when the process template has any global dataslots defined.

Process instance tables
Business Process Server stores the process instances in the PROCESSINSTANCE table\(^\text{16}\), as described below:

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROCESS_TEMPLATE_ID</td>
<td>NUMBER NOT NULL</td>
<td>Template ID of the process(^\text{17})</td>
</tr>
<tr>
<td>PROCESS_INSTANCE_ID(^\text{18})</td>
<td>NUMBER NOT NULL</td>
<td>Instance number of the process</td>
</tr>
<tr>
<td>PROCESS_INSTANCE_NAME</td>
<td>VARCHAR2(256) NOT NULL</td>
<td>Name of the process instance</td>
</tr>
<tr>
<td>CREATOR</td>
<td>CHAR(32) NOT NULL</td>
<td>Name of the creator of this process instance</td>
</tr>
<tr>
<td>STATUS</td>
<td>CHAR(20) NOT NULL</td>
<td>Status of this process instance, for example PI_CREATED</td>
</tr>
</tbody>
</table>

---
\(^{15}\) Table constraint: PROCESS_XML_PKEY
\(^{16}\) Table constraint: PROCESS_INSTANCE_PKEY
\(^{17}\) The IXPROCESS_INSTANCE_CREATOR index is defined on CREATOR and PROCESS_TEMPLATE_ID. The IXPROCESS_INSTANCE_STATUS index is defined on PROCESS_TEMPLATE_ID and STATUS. The IXPROCESS_INSTANCE_START_TIME index is defined on PROCESS_TEMPLATE_ID and START_TIME.
\(^{18}\) Table primary key
<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRIORITY</td>
<td>CHAR(10)</td>
<td>The priority of this process (restrict priority values to Low</td>
</tr>
<tr>
<td>ESTIMATED_DURATION</td>
<td>NUMBER</td>
<td>The estimated number of seconds to complete the process instance</td>
</tr>
<tr>
<td>START_TIME</td>
<td>TIMESTAMP</td>
<td>Starting time of this process instance</td>
</tr>
<tr>
<td>END_TIME</td>
<td>TIMESTAMP</td>
<td>Time when this process instance ends</td>
</tr>
<tr>
<td>WAIT_TIME</td>
<td>NUMBER</td>
<td>Summation of time from workitem creation till the opening of workitem by end user for the process instance</td>
</tr>
<tr>
<td>WORK_TIME</td>
<td>NUMBER</td>
<td>Summation of time from opening of workitem till completion of the same for the process instance</td>
</tr>
<tr>
<td>TOTAL_COST</td>
<td>NUMBER</td>
<td>Summation of cost of performing workitem for the process instance</td>
</tr>
<tr>
<td>DUE_DATE</td>
<td>TIMESTAMP</td>
<td>The date on which the process instance is due</td>
</tr>
<tr>
<td>DURATION</td>
<td>DURATION</td>
<td>The duration of the process instance</td>
</tr>
</tbody>
</table>

**Meta data information tables**

Business Process Server also creates a set of tables for storing meta data information about process instances and worksteps. These tables are defined as follows:

**PROCESSDATAINFO table**

The PROCESSDATAINFO table is defined as follows:

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROCESS TEMPLATE ID</td>
<td>NUMBER NOT NULL</td>
<td>The unique template ID for a process</td>
</tr>
<tr>
<td>DATASLOT NAME</td>
<td>VARCHAR2(32) NOT NULL</td>
<td>Name of the dataslot presented in Designer</td>
</tr>
<tr>
<td>DATASLOT TYPE</td>
<td>NUMBER NOT NULL</td>
<td>JDBC type represented as an integer value</td>
</tr>
</tbody>
</table>

19 Table constraint: PROCESS_DATA_INFO_PKEY
20 Table primary key
The BPServer datatype VARCHAR2(16) NOT NULL

Access permission for the Management module of Business Process Portal CHAR(1) BIZMANAGE_ACCESS_FLAG

The scale if there is a decimal dataslot NUMBER SCALE

Where the dataslot is either global or not CHAR(1) GLOBAL_FLAG

The default size is 128 characters. If you expect the dataslot label size to exceed this number, then you may alter the database schema for this field label. VARCHAR2(32) LABEL

Not used currently. CHAR(1) BIZSITE_ACCESS_FLAG

Informs if this dataslot is the External Instance ID for monitoring worksteps NUMBER(1) MONITOR_ID

PROCESSWORKSTEPINFO table

The PROCESSWORKSTEPINFO table is defined as follows:

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROCESS_TEMPLATE_ID</td>
<td>NUMBER NOT NULL</td>
<td>Unique template ID for a process</td>
</tr>
<tr>
<td>WORKSTEP_NAME</td>
<td>VARCHAR2(32) NOT NULL</td>
<td>Name of the workstep</td>
</tr>
<tr>
<td>WORKSTEP_TYPE</td>
<td>CHAR(15) NOT NULL</td>
<td>Type of workstep, for example: nestedWS</td>
</tr>
<tr>
<td>MILESTONE</td>
<td>VARCHAR2(256)</td>
<td>The milestone workstep information</td>
</tr>
<tr>
<td>WORKSTEP_ID</td>
<td>NUMBER</td>
<td>The ID of the workstep</td>
</tr>
<tr>
<td>MILESTONE_NAME</td>
<td>VARCHAR2(64)</td>
<td>The name of the milestone</td>
</tr>
<tr>
<td>MILESTONE_DESC</td>
<td>VARCHAR2(512)</td>
<td>The description of the milestone</td>
</tr>
</tbody>
</table>

---

21 Table constraint: PROCESS_WORKSTEP_INFO_PKEY

OpenEdge Business Process Server: Application Developer's Guide 167
## SUBPROCESSINFO table

Business Process Server uses the SUBPROCESSINFO table to store process and subprocess relationships. This table is defined as follows:

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROCESS_TEMPLATE_ID</td>
<td>NUMBER NOT NULL</td>
<td>Template ID of the process</td>
</tr>
<tr>
<td>PROCESS_INSTANCE_ID</td>
<td>NUMBER NOT NULL</td>
<td>Instance number of the process</td>
</tr>
<tr>
<td>PARENT_PROCESS_INSTANCE_ID</td>
<td>NUMBER NOT NULL</td>
<td>Instance ID of the parent process of this subprocess</td>
</tr>
<tr>
<td>PARENT_WORKSTEP_NAME</td>
<td>VARCHAR2(32) NOT NULL</td>
<td>Name of the parent workstep, which started this subprocess</td>
</tr>
<tr>
<td>PARENT_WORKSTEP_LOOPCOUNTER</td>
<td>NUMBER</td>
<td>The loop counter of the parent subprocess workstep</td>
</tr>
</tbody>
</table>

## Workstep tables

The WORKSTEP table, which stores worksteps, is defined as follows:

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROCESS_TEMPLATE_ID</td>
<td>NUMBER NOT NULL</td>
<td>Template ID of the process</td>
</tr>
<tr>
<td>PROCESS_INSTANCE_ID</td>
<td>NUMBER NOT NULL</td>
<td>Instance number of the process</td>
</tr>
<tr>
<td>WORKSTEP_NAME</td>
<td>VARCHAR2(32) NOT NULL</td>
<td>Name of the workstep</td>
</tr>
<tr>
<td>PERFORMER</td>
<td>VARCHAR2(64)</td>
<td>Name of the performer of this workstep</td>
</tr>
<tr>
<td>GROUP_TYPE</td>
<td>CHAR(3)</td>
<td>How the actual performer of this workstep is decided, for example ALL, ANY, or NONE.</td>
</tr>
</tbody>
</table>

---

22 Table constraint: SUB_PROCESS_PKEY
23 The IXSUB_PROCESS_INFO_PARENT index is defined on PARENT_PROCESS_INSTANCE_ID and PARENT_WORKSTEP_NAME.
24 Table constraint: WORKSTEP_PKEY
25 Table primary key
26 The IXWORKSTEP_STATUS index is defined on PROCESS_TEMPLATE_ID and STATUS.
<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOOPCOUNTER</td>
<td>NUMBER</td>
<td>The number of times this workstep runs.</td>
</tr>
<tr>
<td>STATUS</td>
<td>CHAR(20)</td>
<td>Status of the workstep</td>
</tr>
<tr>
<td>PRIORITY</td>
<td>CHAR(10)</td>
<td>User's preferred choice of priority (restrict priority values to Low</td>
</tr>
<tr>
<td>ESTIMATED_DURATION</td>
<td>NUMBER</td>
<td>Estimated time in seconds to complete the workstep</td>
</tr>
<tr>
<td>START_TIME</td>
<td>DATE</td>
<td>Starting time of the workstep</td>
</tr>
<tr>
<td>END_TIME</td>
<td>DATE</td>
<td>Time when the workstep ends</td>
</tr>
<tr>
<td>TYPE</td>
<td>VARCHAR2(15)</td>
<td>The type of the workstep</td>
</tr>
<tr>
<td>WAIT_TIME</td>
<td>NUMBER</td>
<td>Summation of time from workitem creation till the opening of workitem by end user for the workstep</td>
</tr>
<tr>
<td>WORK_TIME</td>
<td>NUMBER</td>
<td>Summation of time from opening of workitem till completion of the same for the workstep</td>
</tr>
<tr>
<td>TOTAL_COST</td>
<td>NUMBER</td>
<td>Summation of cost of performing workitem for the workstep</td>
</tr>
<tr>
<td>DUE_DATE</td>
<td>TIMESTAMP</td>
<td>The date on which the workstep is due</td>
</tr>
<tr>
<td>DURATION</td>
<td>NUMBER</td>
<td>The duration of the workstep</td>
</tr>
<tr>
<td>INLINEBLOCK_NAME</td>
<td>VARCHAR2(32)</td>
<td>The inline subprocess name to which this workstep belongs to</td>
</tr>
<tr>
<td>MONITOR_STEP</td>
<td>NUMBER(1)</td>
<td>Informs if this is monitoring workstep</td>
</tr>
</tbody>
</table>

In this table, the "performer" field always contains the latest value, even if the workstep is assigned to another performer/assignee. The "performer" field contains the actual group_name for group task.

When a workstep of type GROUP-BY-ANY is activated, BP Server can temporarily assign the literal value ANY as the performer name. When a performer is then assigned to this workstep later, BPM Process Store updates the field with the actual assignee name.

**Note:** This approach modifies the primary key and might therefore affect performance.
Work item tables

The WORKITEM table fields, which stores workitems, is defined as follows:

**WORKITEM table fields**

The WORKITEM table is defined as follows:

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORKITEM_ID</td>
<td>NUMBER NOT NULL</td>
<td>Identity of the work item</td>
</tr>
<tr>
<td>PROCESS_TEMPLATE_ID</td>
<td>NUMBER NOT NULL</td>
<td>Template ID of the process</td>
</tr>
<tr>
<td>PROCESS_INSTANCE_ID</td>
<td>NUMBER NOT NULL</td>
<td>Instance number of the process</td>
</tr>
<tr>
<td>WORKSTEP_NAME</td>
<td>VARCHAR2(32) NOT NULL</td>
<td>Name of the workstep</td>
</tr>
<tr>
<td>PERFORMER</td>
<td>VARCHAR2(64) NOT NULL</td>
<td>Name of the performer of this work item</td>
</tr>
<tr>
<td>LOOPCOUNTER</td>
<td>NUMBER NOT NULL</td>
<td>The number of times this workstep runs.</td>
</tr>
<tr>
<td>STATUS</td>
<td>CHAR(20)</td>
<td>Status of the work item/performer</td>
</tr>
<tr>
<td>PRIORITY</td>
<td>CHAR(10)</td>
<td>Priority of the work item</td>
</tr>
<tr>
<td>ESTIMATED_DURATION</td>
<td>NUMBER</td>
<td>Estimated time in seconds to complete the work item</td>
</tr>
<tr>
<td>START_TIME</td>
<td>DATE</td>
<td>Starting time of the work item</td>
</tr>
<tr>
<td>END_TIME</td>
<td>DATE</td>
<td>Time when the work item ends</td>
</tr>
</tbody>
</table>

**Note:** If infopad requires higher dimensions, then the higher dimension index columns is also generated.

Dashboard database tables

Dashboard presents the status and progressive information of enterprise applications graphically. It uses database tables to store statistical information that is displayed in widgets.

Business Process Server maintains the following five system tables for the default Dashboard pages:

- STAT_APP_AVG_DURATION table

---

27 Table constraint: WORKITEM_PKEY
28 Table primary key
29 CREATE INDEX IXWORKITEM_STATUS ON work item
30 CREATE INDEX ixworkitem ON work item
Dashboard database tables

- STAT_APP_PRIORITY table
- STAT_APP_DUE_INFO table
- STAT_APP_STATUS_HISTORY table
- STAT_WI_DUE_INFO table

These tables are created when you start the Dashboard for the first time. They refresh periodically when the Dashboard is running. They are removed when you run the `setupSBM` utility.

**STAT_APP_AVG_DURATION table**

This table stores the average time (in hours) to complete an application. Each different application has a row in the following table.

**STAT_APP_AVG_DURATION table fields**

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROCESS_TEMPLATE_NAME</td>
<td>VARCHAR2(64) NOT NULL</td>
<td>The process template name</td>
</tr>
<tr>
<td>PROCESS_TEMPLATE_ID</td>
<td>NUMBER NOT NULL</td>
<td>Unique template ID, 0 ID covers statistics over all applications.</td>
</tr>
<tr>
<td>AVERAGE_TIME</td>
<td>NUMBER NOT NULL</td>
<td>Average application completed duration</td>
</tr>
</tbody>
</table>

**STAT_APP_PRIORITY table**

This table stores the number of applications with different priorities.

**STAT_APP_PRIORITY table fields**

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROCESS_TEMPLATE_NAME</td>
<td>VARCHAR2(64)</td>
<td>The process template name</td>
</tr>
<tr>
<td>PROCESS_TEMPLATE_ID</td>
<td>NUMBER NOT NULL</td>
<td>Process template ID, 0 ID covers statistics over all applications.</td>
</tr>
<tr>
<td>CRITICAL</td>
<td>NUMBER NOT NULL</td>
<td>Number of critical priority process instances</td>
</tr>
<tr>
<td>HIGH</td>
<td>NUMBER NOT NULL</td>
<td>Number of high priority process instances</td>
</tr>
</tbody>
</table>

31 Table primary key. If it is 0, then the statistics are collected over all applications. The corresponding PROCESS_TEMPLATE_NAME is "SBM_ALL_APP."

32 Table primary key. If 0, then the statistics are collected over all applications. The corresponding PROCESS_TEMPLATE_NAME is "SBM_ALL_APP."
**STAT_APP_DUE_INFO table**

This table stores information such as how many applications are currently active. It also indicates how many applications are overdue or will be due by the end of the day, so that you can monitor those applications.

**STAT_APP_DUE_INFO table fields**

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROCESS_TEMPLATE_NAME</td>
<td>VARCHAR2(64)</td>
<td>The process template name</td>
</tr>
<tr>
<td>PROCESS_TEMPLATE_ID</td>
<td>NUMBER NOT NULL</td>
<td>Unique template ID, 0 ID covers statistics over all applications.</td>
</tr>
<tr>
<td>ACTIVE</td>
<td>NUMBER NOT NULL</td>
<td>Number of active process instances</td>
</tr>
<tr>
<td>OVERDUE</td>
<td>NUMBER NOT NULL</td>
<td>Number of overdue process instances</td>
</tr>
<tr>
<td>DUE_TODAY</td>
<td>NUMBER NOT NULL</td>
<td>Number of process instances due today</td>
</tr>
</tbody>
</table>

**STAT_APP_STATUS_HISTORY table**

From this table, you can see how many applications were activated, completed, removed, or suspended for the last several days.

**STAT_APP_STATUS_HISTORY table fields**

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROCESS_TEMPLATE_NAME</td>
<td>VARCHAR2(64) NOT NULL</td>
<td>Process template name</td>
</tr>
<tr>
<td>PROCESS_TEMPLATE_ID</td>
<td>NUMBER NOT NULL</td>
<td>Unique template ID, 0 ID covers statistics over all applications.</td>
</tr>
<tr>
<td>ACTIVATED</td>
<td>NUMBER NOT NULL</td>
<td>Number of activated process instances every day</td>
</tr>
</tbody>
</table>

---

33 Table primary key. If 0, then the statistics are collected overall applications. The corresponding PROCESS_TEMPLATE_NAME is "SBM_ALL_APP."

34 Table primary key. If 0, then the statistics are collected overall applications. The corresponding PROCESS_TEMPLATE_NAME is "SBM_ALL_APP."
**Dashboard database tables**

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMPLETED</td>
<td>NUMBER NOT NULL</td>
<td>Number of completed process instances every day</td>
</tr>
<tr>
<td>REMOVED</td>
<td>NUMBER NOT NULL</td>
<td>Number of removed process instances every day</td>
</tr>
<tr>
<td>SUSPENDED</td>
<td>NUMBER NOT NULL</td>
<td>Number of suspended process instances every day</td>
</tr>
<tr>
<td>RECORD_TIME³⁵</td>
<td>DATE NOT NULL</td>
<td>Time updating this record</td>
</tr>
</tbody>
</table>

**STAT_WI_DUE_INFO table**

This table is similar to the STAT_APP_DUE_INFO table. The only difference is that it stores information on application worksteps instead of the entire applications.

**STAT_WI_DUE_INFO table fields**

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROCESS_TEMPLATE_NAME</td>
<td>VARCHAR2(64) NOT NULL</td>
<td>Process template name.</td>
</tr>
<tr>
<td>PROCESS_TEMPLATE_ID³⁶</td>
<td>NUMBER NOT NULL</td>
<td>Unique template ID, 0 ID covers statistics over all applications.</td>
</tr>
<tr>
<td>PERFORMER³⁷</td>
<td>VARCHAR2(64)</td>
<td>Workstep performer.</td>
</tr>
<tr>
<td>ACTIVE</td>
<td>NUMBER NOT NULL</td>
<td>Number of active worksteps.</td>
</tr>
<tr>
<td>OVERDUE</td>
<td>NUMBER NOT NULL</td>
<td>Number of overdue worksteps.</td>
</tr>
<tr>
<td>DUE_TODAY</td>
<td>NUMBER NOT NULL</td>
<td>Number of worksteps due today.</td>
</tr>
</tbody>
</table>

³⁵ Table primary key
³⁶ Table primary key. If it is 0, then the statistics are collected over all applications. The corresponding PROCESS_TEMPLATE_NAME is "SBM_ALL_APP."
³⁷ Table primary key
Developing management applications to control infopads

Business Process Server offers Business Process Portal users an application framework for developing management applications to control infopad data. Typically, Business Process Server application data is stored in application-specific infopads as well as in user-defined dataslots in the process repository. You can view and update user-defined dataslot values using the Process Status Viewer in the Management module of Business Process Portal, and can change infopad slot values in the Infopads page of the Management module. You can use the information in this chapter to create advanced management applications that enable you to control and manipulate infopad data.

Note: For basic information on creating and editing the infopad and its properties, refer the OpenEdge Getting Started: Developing BPM Applications with Developer Studio. For basic information on changing infopad slot values, "Using the Infopad Manager" in the Business Process Portal Manager’s Guide.

The framework and approach described in this chapter can be used to reduce the time and effort taken to create these infopad management applications. The framework consists of the following components:

• Creating a custom HTML application on page 176 to manipulate infopad data using only HTML.
• Creating a Java applet on page 178 to develop an interface for controlling infopad data in a Java applet.
Why create a management application? Imagine the following scenario. You have a purchasing process that requires every purchase request over $1500 to secure a secondary authorization. However, due to an acquisition, your company’s policy changes so that only purchases over $2000 require secondary authorization. Instead of hard-coding this value in BP Server, you might store the value in an infopad. Using the management application framework, you can easily create a custom application to allow managers to change this and other infopad values.

For details, see the following topics:

- Creating a custom HTML application
- Creating a Java applet
- Parameter values for applet HTML file definitions

Creating a custom HTML application

Creating a management application using a custom HTML page is perhaps the easiest way to generate an interface allowing you to manipulate infopad values. The basic procedure involves creating a regular form-based HTML page with the necessary user interface components required to display and update information. Once this is in place, you can use custom HTML tags to display, get, and update the actual infopad values.

When a user requests this page, Business Process Portal preprocesses the HTML file substituting JavaScript and conventional HTML in place of the custom tags, generating a page that can display and update infopad values suitable for display on any supported web browser.

To create the custom HTML page, complete the following steps:

1. Create the HTML file with the user interface components required to display and update the infopad values.
   - Use standard HTML constructs including input fields, drop-down lists, and multiline inputs (textarea).

2. Use custom HTML tags to display, get, store, and update infopad values.
   - The following table describes the custom tags you can use:

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;InfopadValue NewTable x y slot&gt;</td>
<td>Causes a variable to be substituted for the infopad value allowing the user to view and modify the value.</td>
</tr>
<tr>
<td>&lt;InfopadGetValue NewTable x y slot&gt;</td>
<td>Causes the value of the dataslot to be substituted. No update of the value is possible.</td>
</tr>
<tr>
<td>&lt;InfopadStoreValue NewTable x y slot&gt;</td>
<td>Specifies the dataslot to update. Useful in HTML constructs such as SELECT.</td>
</tr>
<tr>
<td>&lt;InfopadUpdate button_label&gt;</td>
<td>Specifies the button label along with the action to perform once the button is pressed.</td>
</tr>
</tbody>
</table>

3. Save the HTML file.

4. Update Business Process Portal user’s application file to include information about the new application.
The following is an example of a customized HTML page illustrating the use of the special tags:

```html
<html>
<head>
</head>
<BODY>
<form>
<INPUT Type='text' VALUE='InfopadValue NewTable 5 5 slot1'>
<INPUT Type='text' VALUE='InfopadValue NewTable 4 4 count'>
<p></p>
<Select <InfopadStoreValue NewTable row5 col5 slot5>>
<option value='value1'><InfopadGetValue NewTable row4 4 slot6>
<option VALUE='value2'><InfopadGetValue NewTable row5 5 slot6>
<option value='value3'><InfopadGetValue NewTable row4 4 slot5>
<option> <InfopadGetValue NewTable row1 col4 slot2>
</select>
<p></p>
<InfopadUpdate Update>
</FORM>
</BODY>
</html>
```

The following shows the same HTML file after translation by the servlet, usable on any Business Process Server-supported web browser:

```html
<html>
<head>
</head>
<SCRIPT LANGUAGE='JavaScript'>
function returnfalse()
{ return false;
}function handleform0()
{
 var newLocation = "http://machine:80/BPM Manage API//com.tdiinc.BPM Manage API t.Servlet.ApplicationServlet?serviceType=postApplication&form=form0&SessionID=3fe3fb26d5198b5d.1.947286386593";
 newLocation += "&element0=document.form0.element0.value; 
 newLocation += "&element1=document.form0.element1.value; 
 newLocation += "&element2=document.form0.element2.options[document.form0. element2.selectedIndex].value; 
 location = newLocation;
}</script>
<body>
<form name='form0' onsubmit='returnfalse()'>
<input type='text' value='15' name='element0'>
<input type='text' value='10' name='element1'>
<p></p>
<select name='element2'>
<option value='value1'>howard<option value='value2'>howard<option value='value3'>gary<option>
25</select>
<p></p>
<input type='button' name='updatebutton' onclick='handleform0()' value='Update'>
</form>
</body>
</html>
```

You must create a directory to store your HTML application.
Creating a Java applet

You can use a Java applet to create an interface for manipulating infopad values. Using a Java applet instead of a customized HTML page offers you greater flexibility in terms of user interface and general processing capabilities, all in exchange for a small increase in application complexity.

Normally, you use an HTML page to load the applet. When the user requests the page containing the Java applet, Business Process Portal displays the interface enabling users to view and modify the infopad values.

To create a Java applet allowing infopad manipulation, complete the following steps:

1. Create the Java applet with the user interface components required to display and update the infopad values.
2. Include the commands to request, set, and update the infopad values.

The following table describes the commands you can use:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>For example, see . . .</th>
</tr>
</thead>
<tbody>
<tr>
<td>setInfopadRequest</td>
<td>Instructs the servlet to accept a vector from the applet, saving the vector as part of the user’s session information. The servlet later uses this information to query BPM Events for the infopad values.</td>
<td>Using the setInfopadRequest service on page 179.</td>
</tr>
<tr>
<td>requestInfopadValues</td>
<td>Retrieves the request vector from the user’s session information, and queries BPM Events for the required values. The servlet then packages the results into a vector and sends the result to the applet.</td>
<td>Using the requestInfopadValues service on page 180.</td>
</tr>
<tr>
<td>updateInfopad</td>
<td>Posts a vector of updated infopad values to the servlet. The servlet converts the values in the vector to events and sends the events to BPM Events to update the required values.</td>
<td>Using the updateInfopad service on page 180.</td>
</tr>
</tbody>
</table>

**Important:** Use the Integer, Character, and Float object wrappers for the corresponding primitive types of int, char, and float. For example, if you want to pass an integer value to the servlet, then use a method signature of integer and use the corresponding object wrapper for the primitive.

3. Save the applet.
4. Update Business Process Portal user’s application file to include information about the new application.
Using the setInfopadRequest service

To send infopad requests to the servlet, you may request values from infopads in a granularity of a single slot, or an infopad cell. Requests are grouped together into a Vector for performance reasons. Due to this, even if you only want the values from one infopad cell or slot, then you must package it in a Vector. Then for each request you want to send, encapsulate it in a Hashtable. The name of the infopad you want to query is stored with the key "Tablename" and the value is the actual name of the infopad. The row you want to query is stored with the key "Row" and the value is either the row name or the row number. The column you want to query is stored with the key "Column" and the value is either the column name or the column number. The slot whose value you want to return is packaged with the key "Slot" and the value is the slot name. If you want to have the entire cell returned, then use the key "Slot" and the value is ".*".

Once you have packaged your requests, you may open a connection to the servlet, and send the Vector. You can send the request to the following URL:

```java
<hostname>BPM Manage API/com.tdiinc.BPM Manage APIr.ApplicationServlet.ApplicationServlet?serviceType=setInfopadRequest
```

**Note:** Remember to append the registration string to the URL.

The variable `registrationString` contains the sessionID inserted into one of the applet's parameter tags. You can then use the `getParameter`(" ") method to get the actual value, passing it the name of the parameter containing the sessionID. Then you can use the `requestInfopadValues` service to retrieve the results.

The following example of using the `setInfopadRequest` service shows how the "/&" is used to connect the registrationString to the URL.

```java
Vector updates = new Vector();
Hashtable newValues = new Hashtable();
newValues.put("Tablename","NewTable");
newValues.put("Row","1");
newValues.put("Column","1");
newValues.put("Slot","count");
updates.addElement(newValues);
newValues = new Hashtable();
newValues.put("Tablename","NewTable");
newValues.put("Row","2");
newValues.put("Column","5");
newValues.put("Slot","slot1");
updates.addElement(newValues);
newValues = new Hashtable();
newValues.put("Tablename","NewTable");
newValues.put("Row","3");
newValues.put("Column","2");
newValues.put("Slot","slot2");
updates.addElement(newValues);
URL url = new URL("http://hawaii/BPM Manage API/com.tdiinc.BPM Manage APIr.ApplicationServlet.ApplicationServlet?serviceType=setInfopadRequest&" + registrationString); URLConnection con = url.openConnection(); con.setDoInput(true);
con.setDoOutput(true);
con.setRequestProperty("Content-Type","application/octet-stream");
ObjectOutputStream out = new ObjectOutputStream(con.getOutputStream()); out.writeObject(updates); out.flush();
out.close();
ObjectInputStream inputFromServlet = new ObjectInputStream(con.getInputStream());
inputFromServlet.close();
```
Using the requestInfopadValues service

Once you have sent the request for infopad values to the servlet, you can call the servlet to request the infopad values. You can send the request to the following URL:

Note: Remember to append the registration string to the URL.

The variable registrationString contains the sessionID inserted into one of the applet's parameter tags. You can then use the getParameter (" ") method to get the actual value, passing the name of the parameter containing the sessionID. You may then read the input object from the ObjectInputStream. The object passed back to the applet is a Vector. The responses in the Vector correlate with requests, meaning the responses appear in the Vector at the same location as the corresponding request. Each response is a Hashtable. The infopad the response was acquired from is stored in the key "Tablename," the row name is stored in the key "RowName," the column name is stored in the key "ColName," the row number is stored in the key "RowNumber," and the column number is stored in the key "ColNumber." All other values in the Hashtable are slot name-value pairs. They cannot be enumerated here as they depend on the actual slots defined in the infopad.

The following example of using the requestInfopadValues service shows how the "&" is used to connect the registrationString to the URL.

Using the updateInfopad service

To send infopad updates to the servlet, you may update values from infopads in a granularity of a single slot, or an infopad cell. Updates are grouped together into a Vector for performance reasons, even if you only want to update the values from one infopad cell or slot, then for each update you want to send, encapsulate it in a Hashtable. The name of the infopad you want to update is stored with the key "Tablename" and the value is the actual name of the infopad. The row you want to update is stored with the key "Row" and the value is either the row name or the row number. The column you want to update is stored with the key "Column" and the value is either the column name or the column number. The slot whose value you want to return should be packaged with the slot name as the key and the updated value as the value. Only those slots that you want updated, should be placed into the Hashtable. Once you have packaged your updates, you may open a connection to the servlet, and send the Vector.

You can send the update request to the following URL:

Note: Remember to append the registration string to the URL.

The following example shows how the "&" is used to connect the registrationString to the URL. The variable registrationString contains the sessionID inserted into one of the applet's parameter tags. You can then use the getParameter (" ") method to get the actual value, passing it the name of the parameter containing the sessionID.

The following is an example of how to use the updateInfopad service:

```java
Vector updates = new Vector();
for (int i=1;i<6;i++)
    for (int j=1;j<6;j++)
        Hashtable firstupdate = new Hashtable();
        firstupdate.put("Tablename","NewTable");
        firstupdate.put("Row",String.valueOf(i));
        firstupdate.put("Column",String.valueOf(j));
        firstupdate.put("count","10");
        firstupdate.put("slot1","15");
        firstupdate.put("slot2","25");
```
Parameter values for applet HTML file definitions

Four parameter values are applicable in applet HTML file definitions.

Parameter values

- `<%applicationDir%>`. A shortcut to the applications directory OEBPS_HOME\BPM Manage API\root\applications. It replaces the value with the actual directory location.

- `<%hostname%>`. A placeholder for the hostname and port number of the server machine hosting Business Process Portal. It replaces the value with the actual hostname and port number.

- `<%bizPulseHost%>`. A placeholder for the hostname of the server which is hosting BPM Events. It replaces the value with the actual hostname of the machine hosting BPM Events. If BPM Events is running on the same computer as Business Process Portal, then the replacement value is blank. If this occurs, then use a local host instead.

- `<%sessionID%>`. A placeholder for the user’s session ID. It replaces the value with the current run-time session ID. You can append this ID to servlet calls from the applet, otherwise a security exception is thrown and the user is required to log back in.

Configuring BPM Events

Two configuration parameters are related to the management application framework, but you must only concerned with the parameters when the BPM Events server that you are using does not reside on the local computer. If the BPM Events server is on another computer, then use the Business Process Portal to configure the BPM Events host machine and BPM Events server parameters on the Business Process Portal Management module page.

Note: If the BPM Events server resides on the local computer, then Business Process Portal collects the configuration information from the bpmevents.conf file.
Developing Balanced Scorecard applications

Balanced Scorecard is one of the premier performance management approaches for translating an enterprise and organizational vision into a set of measurable objectives. These objectives help organizations set targets and monitor performance.

Business Process Server has developed the Balanced Scorecard application as a business process that performs data collection, computes the Key Performance Indicators (KPIs), and monitors the performance data sources on a scheduled basis. Using Business Process Portal Management module, users can change the performance targets and update the performance thresholds, worst and best values, and KPI relative weights; view KPI scores in a dashboard-like interface with selected indicators; and manage Balanced Scorecard applications.

Business Process Server is best suited for defining and implementing the Balanced Scorecard management process. This process includes defining KPI measures and performance targets, integrating with external systems to collect performance data, and providing a Balanced Scorecard interface for ease in monitoring KPIs. Additionally, business managers can analyze KPI relevant data through Business Process Portal and make proactive performance decisions that allow them to meet the performance targets set by the organization.

This chapter presents an overview of Balanced Scorecard, a discussion of the Balanced Scorecard application, and a description of the Balanced Scorecard rule file syntax.

For basic information on viewing or modifying existing Balanced Scorecard applications, or designing new Balanced Scorecards, see Chapter 9, "Using the Balanced Scorecard" in the Business Process Portal Manager’s Guide.

For details, see the following topics:

- Understanding the Balanced Scorecard
- Balanced scorecard rules
- Balanced scorecard infopads
Understanding the Balanced Scorecard

The Balanced Scorecard methodology provides a performance management approach for organizations. By definition, the Balanced Scorecard approach provides a high-level view of enterprise accounting for all business areas. Using this approach, top level executives can assess the company's potential to reach the set targets and take note of any changes in performance.

In addition, Balanced Scorecard enables clear communication between top-level management and middle-level management. Realization of this approach requires knowledge of the following:

• Identification of key performance indicators (KPIs). You can use standard, predefined KPIs or the Balanced Scorecard Designer to create customized KPIs. For more information on using Balanced Scorecard and on defining Balanced Scorecard applications, see Business Process Portal Manager’s Guide.

• Defining a process for monitoring and collecting relevant data from different data sources in the organization.

• Along with proper monitoring of KPIs, you can use detailed KPIs to measure the contribution business processes are making in achieving business objectives and corporate strategies.

Focus on four perspectives

The central idea of a Balanced Scorecard is to link a measurement system to your organization’s strategic goal. Typically, Balanced Scorecard uses the following four measurement systems, or Perspectives, to indicate how an organization is meeting its strategic goals:

• Finance — Performance measures include such key performance indicators (KPIs) as product and service revenues, revenue growth, and profit margins.

• Customer relations — Performance measures include such KPIs as customer retention rate, customer satisfaction level, product returns, service response time, and service contract renewals.

• Internal processes — KPIs that measure performance include personnel retention, leave of absence or sickness, delivery on time, average recruitment time, discontinued products or projects, and the average help desk service time.

• Innovation — Performance is measured by KPIs that include number of patents received/filed, and the number of excellence awards received, the number technical papers published, the average number of training days per employee, and the percentage of employees promoted.

The Balanced Scorecard application (see the Reviewing the balanced scorecard application topic) provided with Business Process Server uses these perspectives.

Key performance indicators (KPIs)

Key performance indicators (KPIs) are the basis for Balanced Scorecards. In each perspective, you must identify unique KPIs. KPI progress can be monitored on a daily, weekly, monthly, yearly, or other basis. For example, for monthly monitoring, the current value of the indicator is collected monthly and the score is updated accordingly.

KPI examples

Examples are provided in the following table.
<table>
<thead>
<tr>
<th>Perspective</th>
<th>Possible KPIs</th>
<th>Worst value</th>
<th>Best value</th>
<th>Current value</th>
<th>Score (0-10)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Finance</strong></td>
<td>Product revenue</td>
<td>$125,000</td>
<td>$1,500,000</td>
<td>$250,000</td>
<td>0.90</td>
</tr>
<tr>
<td></td>
<td>Service revenue</td>
<td>$375,000</td>
<td>$2,500,000</td>
<td>$750,000</td>
<td>1.76</td>
</tr>
<tr>
<td></td>
<td>Product revenue growth</td>
<td>5%</td>
<td>35%</td>
<td>$750,000</td>
<td>0.90</td>
</tr>
<tr>
<td></td>
<td>Service revenue growth</td>
<td>10%</td>
<td>35%</td>
<td>11%</td>
<td>2.00</td>
</tr>
<tr>
<td></td>
<td>Profit margin</td>
<td>$80,000</td>
<td>$800,000</td>
<td>$93,000</td>
<td>0.18</td>
</tr>
<tr>
<td></td>
<td>Sales pipeline</td>
<td>$1,000,000</td>
<td>$20,000,000</td>
<td>$3,000,000</td>
<td>1.05</td>
</tr>
<tr>
<td><strong>Customer Relations</strong></td>
<td>Customer retention rate</td>
<td>50%</td>
<td>100%</td>
<td>80%</td>
<td>6.00</td>
</tr>
<tr>
<td></td>
<td>Customer satisfaction level</td>
<td>75%</td>
<td>100%</td>
<td>80%</td>
<td>2.00</td>
</tr>
<tr>
<td></td>
<td>Product return as a percentage of sales</td>
<td>15%</td>
<td>0%</td>
<td>7.5%</td>
<td>5.00</td>
</tr>
<tr>
<td></td>
<td>Service response time</td>
<td>20 days</td>
<td>1 day</td>
<td>4 days</td>
<td>8.42</td>
</tr>
<tr>
<td></td>
<td>Service contract renewals as a percent of service contract agreements</td>
<td>50%</td>
<td>90%</td>
<td>75%</td>
<td>6.25</td>
</tr>
<tr>
<td><strong>Internal processes</strong></td>
<td>Personnel retention rate</td>
<td>75%</td>
<td>10%</td>
<td>85%</td>
<td>4.00</td>
</tr>
<tr>
<td></td>
<td>Avg. number of sick days</td>
<td>15 days</td>
<td>0 days</td>
<td>3 days</td>
<td>8.00</td>
</tr>
<tr>
<td></td>
<td>Product/Development/Project/Sales/Marketing delivery on time (%)</td>
<td>50%</td>
<td>100%</td>
<td>85%</td>
<td>7.00</td>
</tr>
<tr>
<td></td>
<td>Avg. recruitment time</td>
<td>30 days</td>
<td>1 day</td>
<td>15 days</td>
<td>5.17</td>
</tr>
<tr>
<td></td>
<td>Funds spent on discontinued projects/products</td>
<td>$5,000,000</td>
<td>$100,000</td>
<td>$1,000,000</td>
<td>8.01</td>
</tr>
<tr>
<td></td>
<td>Average Help Desk Service Time</td>
<td>10 days</td>
<td>1 day</td>
<td>3 days</td>
<td>7.78</td>
</tr>
<tr>
<td>Perspective</td>
<td>Possible KPIs</td>
<td>Worst value</td>
<td>Best value</td>
<td>Current value</td>
<td>Score (0-10)</td>
</tr>
<tr>
<td>---------------------</td>
<td>--------------------------------------------------</td>
<td>-------------</td>
<td>------------</td>
<td>---------------</td>
<td>--------------</td>
</tr>
<tr>
<td>Innovation</td>
<td>Number of patents received/filed</td>
<td>0</td>
<td>4</td>
<td>1</td>
<td>2.50</td>
</tr>
<tr>
<td></td>
<td>Number of excellence awards received</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td>Number of technical papers/books published</td>
<td>0</td>
<td>5</td>
<td>2</td>
<td>4.00</td>
</tr>
<tr>
<td></td>
<td>Percentage of employees promoted</td>
<td>5%</td>
<td>15%</td>
<td>7%</td>
<td>2.00</td>
</tr>
<tr>
<td></td>
<td>Average number of training days per employee</td>
<td>0 days</td>
<td>10 days</td>
<td>5 days</td>
<td>5.00</td>
</tr>
</tbody>
</table>

### Estimating relative importance of KPIs

The selected key performance indicators (KPIs) in each of the perspectives might not have equal importance. The relative importance of the KPIs vary depending on the market conditions and organizational factors.
Relative importance can also be defined at the high level. For example, the organization can give higher preference to both Finance and Customer Relations KPIs compared to Internal Processes and Innovation. This is detailed in the following figure, and described in more detail in the Business Process Portal Manager’s Guide.

Figure 30: Examples of relative importance

Performance data collection and monitoring

After identifying KPIs and their relative importance, plan the data collection from disparate systems across the organization, and select monitoring systems that periodically collect and transform data into KPI values.

Note: If the organization has OLAP (OnLine Analytical Processing) systems that collect analysis data, then you must develop an OLAP system application to collect KPI corresponding data in each of the areas.

Balanced scorecard rules

Balanced Scorecard rules in BPM Events compute the KPI values from the corresponding dataslot values in the respective Balanced Scorecard processes. In addition, you can schedule rules to periodically start the Balanced Scorecard processes and collect data. Once the KPI data is consolidated into infopads, you can view all the KPI values in Business Process Portal Management module. For more information about viewing the Balanced Scorecard application in the Business Process Portal, see the Business Process Portal Manager’s Guide.
Balanced scorecard infopads

The Balanced Scorecard application contains four two-dimensional infopads: Finance, Customer, Internal, and Innovation/Growth. First, you must define the infopads with the proper rule file syntax. Each infopad is defined for each of the perspectives to hold the KPI values along with worst value, best value, and score slots. One dimension represents the KPIs and another dimension represents the time-axis, for example, months. Each cell in the InfoPad contains 'WorstValue', 'BestValue', 'CurrentValue', 'RelativeWeight', and 'Score' slots. Whenever a business manager changes 'WorstValue' or 'BestValue,' the corresponding score values are updated automatically using an update rule.

Note: For detailed information about Rules and infopad syntax, see the BPM Events User’s Guide.

Defining rule file syntax for balanced scorecard infopads

The following sections are for the advanced user and provide the rule file syntax for the predefined infopads that are derived from Balanced Scorecard applications and are provided in the Business Process Server package. These predefined infopads include the BalancedScorecard, Balanced Scorecardimp, Customer, Finance, Innovation and Internal.

Note: For more information about changing infopad slot values, see "Using the Infopad Manager" in the Business Process Portal Manager's Guide.

BalancedScorecard infopad rule file syntax

The main scorecard, called BalancedScorecard, holds the overall score and is defined as follows.

Syntax

```plaintext
BalancedScorecard := new persistent infopad<cell{Score:float}>[]][1]("BalancedScorecard");
```

This defines the infopad with an unspecified number of rows ([]). One row is created for each time period, usually months. The only slot value needed here is the overall score.

```plaintext
BalancedScorecard.addRowLabels("Months", list{common::common_rules::MonthLookupInfopad[NOW.month()][1].monthName+NOW.year()});
```

This action follows the same row naming convention as the FinanceInfopad and other perspectives.

```plaintext
BalancedScorecard.addColLabels("Scores", list{"Scores"});
```

This adds a label for the column, not very useful in BPM Events but a good feature for the infopad display in other components.

```plaintext
BalancedScorecard[1][1].Score := 7.3969193;
```

This updates the first row with some default value.

```plaintext
schedule("BalancedScorecardRowCreator", (((YEAR:((NOW).year()))/MONTH:((NOW).month()))+(1*MONTH))-(1*DAY)),type:"AddBalancedScorecardScorecardRows",value:"monthly");
```
The last item in the Initialization section is to schedule the event which triggers the rule to add a row to all the necessary infopads.

This line schedules an event to be triggered on the last day of the month the rules were loaded in. The event adds a row to the necessary tables at the end of the month. Please see the BPM Events User’s Guide for additional information on the schedule action.

**BalancedScorecardimp infopad rule file syntax**

You can create a separate infopad to specify the relative weights (or importance) of each of the segments. Update these weights from Business Process Portal. In the following sample, the rule file syntax is presented in code formatted text, immediately followed by the rule explanation.

**Weights**

```plaintext
BalancedScorecardimp := new persistent infopad<cell{Weight:float}>[4][1]("BalancedScorecardimp");
```

Defines the BalancedScorecardimp Infopad which contain the weights of each perspective of the scorecard as it relates to the overall score. In this case, there are four rows, one for each perspective. The cells in this table consist of only the weight of the perspective.

**Rows**

```plaintext
BalancedScorecardimp.addRowLabels("Perspectives", list("Finance", "Customer", "Internal", "Innovation"));
```

Defines the row labels.

**Columns**

```plaintext
BalancedScorecardimp.addColLabels("Weights", list("Current"));
```

Defines the column titles.

**Perspectives**

```plaintext
BalancedScorecardimp["Finance"]["Current"].Weight := 0.3;
BalancedScorecardimp["Customer"]["Current"].Weight := 0.3;
BalancedScorecardimp["Internal"]["Current"].Weight := 0.25;
BalancedScorecardimp["Innovation"]["Current"].Weight := 0.15;
```

The Finance perspective is given the weight of 30%. The Customer Relations perspective is also given the weight of 30%. The Internal Processes perspective is given the weight of 25%. The Innovation perspective is given the weight of 15%. The sum of the weights should add up to one (or 100%).

**Alarm condition**

```plaintext
BalancedScorecardimp.check("updateBalancedScorecardScores", "Weight","NE","slot[Weight]","absolute");
```

The code above defines an alarm condition on the Weight slot in all the cells. An alarm event is sent when the value in the Weight slot is not equal to (NE) the value in the Weight slot. This means, it sends an alarm event when the weight is not equal to the set weight.
Finance, customer, innovation/growth, and internal infopads rule file syntax

The examples provided represent the FinanceInfoPad, but also apply to the Customer, Innovation, and Internal InfoPads. In the following sample, the defined infopad is presented in code formatted text, immediately followed by the infopad explanation.

Syntax

```
Finance := new persistent
infopad<cell{WorstValue:float,BestValue:float,CurrentValue:float,
RelativeWeight:float,Score:float,NegativeRange:string,Description:
string,Unit:string}>[]][?]("Finance");
```

The infopad is defined with an unspecified length in the rows. One row is added for each new time period, typically a month. The value seven signifies the number of KPIs plus one. The extra column is for the sum of the perspective values Score and RelativeWeight.

The infopad contains six custom slots in addition to the default count slot. WorstValue contains the worst-case value for a specific KPI. BestValue contains the value that represents the goal for that KPI. If the company achieves the best value, then it is equal to 10. CurrentValue holds the value collected by the processes. RelativeWeight contains the weight that this KPI contributes to the total perspective score. Again weights should add up to one in the aggregate, but you can normalize later when calculating. Score holds the score for this KPI, calculated from the best, worst, and current values. NegativeRange is either T or F. In the case of T, it indicates that the best and worst values define a negative range. An example of a negative range would be 35 to 4. A KPI which exhibits this would be Employee Turnover where 35% is bad and 4% is good. If NegativeRange is F, then it indicates a normal range such as 5 to 50 percent exhibited by the return on investment KPI.

```
Finance.addColLabels("KPI", list{"Product Revenues", "Service
Revenues", "Product Revenue Growth", "Service Revenue Growth",
"Profit Margin", "Sales Pipeline", "FinanceScore"]);
```

This line defines the column labels. The first value is title for the collective group of column titles, in this case, "KPI." The last value is artificial, and consists of the perspective name, Finance, appended with Score. This represents the column for the totals, that is the overall finance perspective score and relative weight. All of the titles in between are the titles of the individual KPI for the finance perspective.

```
Finance.addRowLabels("Months",
list{common::common_rules::MonthLookupInfopad[NOW.month()]][1].nonth
Name+NOW.year()});
```

This line defines the row label for the first and the only row so far. As the infopad was defined with unspecified row length, it only has one default row. However, you can schedule an event later which adds additional rows at the given time. This line assumes you want to name the rows with the name of the month appended with the year. For example, March2005. A similar naming pattern is followed for additional rows.

```
Finance[1][1].WorstValue := 125000.0;
```

This slot holds the worst-case value for this KPI. If the current value is equal to this value, then the score is 0.

```
Finance[1][1].BestValue := 1500000.0;
```

This slot holds the best value for this KPI. If the current value is equal to this value, then the score is 10.

```
Finance[1][1].CurrentValue := 1250000.0;
```
This slot holds the current value of this KPI. This value is updated from a rule triggered by the periodically-run collection process.

```
Finance[1][1].Unit := "dollars";
Finance[1][1].Description := "Finance - Product Revenues";
```

These slots define the unit and description.

```
Finance[1][1].RelativeWeight := 0.25;
```

This slot holds the weight for this KPI.

```
Finance[1][1].NegativeRange := "F";
```

We recommend you set the negative range explicitly to the correct value for each KPI.

```
Finance[1][1].Score := 8.181818;
```

This slot holds the score for this KPI. This is computed dynamically when the best, worst, or current values or relative weights change.

Same steps are repeated for all the KPIs.

```
Finance[1]["FinanceScore"].BestValue := 1.0;
Finance[1]["FinanceScore"].WorstValue := 0.0;
```

For the Finance perspective, best and worst values are defined, that are used for the graphical display on a dial.

```
Finance[1]["FinanceScore"].Score := 8.151753;
```

This slot holds the score for the perspective. Note that this is the cell in the last column and the last row.

```
Finance[1]["FinanceScore"].RelativeWeight := 1.0;
```

This slot holds the sum of the relative weights of all the KPI in this perspective. This value normalizes the perspective score.

```
Finance.check("worstcheckFinance","WorstValue","NE","slot[WorstValue]","absolute");
```

This action sets an alarm condition which sends an event every time the condition is triggered. In this case, every time the Worst Value is updated, the name is sent with the alarm event. When the event corresponding to this event is sent, the worst value from the FinanceInfopad is modified. You can streamline your rules using this information. Similar shortcuts apply for other alarm generated events as well.

```
Finance.check("bestcheckFinance","BestValue","NE","slot[BestValue]","absolute");
```

This action sends an event every time the BestValue is updated.

```
Finance.check("currentcheckFinance","CurrentValue","NE","slot[CurrentValue]","absolute");
```

This action sends an event every time the CurrentValue is updated.

```
Finance.check("relativecheckFinance","RelativeWeight","NE","slot[RelativeWeight]","absolute");
```
This action sends an event every time the RelativeWeight is updated.

\[
\text{Finance.check("scorecheckFinance","Score","NE","slot[Score]","}
\text{absolute");}
\]

This action sends an event every time the Score is updated for both the KPI and perspective scores.

**Rule file syntax for modifying each scorecard table**

The Finance infopad is an example of how to modify rules for each scorecard table. What applies to the Finance infopad here also applies to the Customer, Innovation, and Internal infopads.

**Syntax**

In the following sample, the rule file syntax is presented in code formatted text, immediately followed by the rule explanation.

```
rule Finance_WorstValue activated by EVT_1 of
BPEVENT_INFOPAD_ALARM::worstcheckFinance

This rule covers updates to the WorstValue slots in the finance infopad. When those values are modified, the alarm condition, worstcheckFinance, is triggered and sends an event. This rule is activated by that event.

then
{Finance[EVT_1.row][EVT_1.column].Score :=
 (((Finance[EVT_1.row][EVT_1.column].CurrentValue - Finance[EVT_1.row][EVT_1.column].WorstValue) /
 (Finance[EVT_1.row][EVT_1.column].BestValue - Finance[EVT_1.row][EVT_1.column].WorstValue))*10)/
 Finance[EVT_1.row]["FinanceScore"].RelativeWeight));

This updates the KPI score of the cell at the row and column where the alarm was triggered, the context.row and context.column provide us with these values.

The current, worst, and best value sub-equation gives the percentage that the current value is between worst value and best value, between 0 and 1. Multiplying by 10 gives us the range of the score from 0 to 10. Dividing by the sum of the relative weights normalizes the score when the sum is greater than one. If the score equals one, then it is inconsequential. This prevents user error from breaking the calculations.

discard(EVT_1);
}
```

This command destroys the event.

```
rule Finance_BestValue activated by EVT_1 of
BPEVENT_INFOPAD_ALARM::bestcheckFinance

then
{Finance[EVT_1.row][EVT_1.column].Score :=
 (((Finance[EVT_1.row][EVT_1.column].CurrentValue - Finance[EVT_1.row][EVT_1.column].WorstValue) /
 (Finance[EVT_1.row][EVT_1.column].BestValue - Finance[EVT_1.row][EVT_1.column].WorstValue))*10)/
 Finance[EVT_1.row]["FinanceScore"].RelativeWeight));

discard(EVT_1);
}
```

This command destroys the event.
This rule does the same action as the worst check rule, only it is triggered by updates to the BestValue slots in the Finance Infopad. When that occurs, it updates the score for that KPI. See the description of the previous rule for more information.

```java
rule Finance_CurrentValue_GT activated by EVT_1 of BPEVENT_INFOPAD_ALARM::currentcheckFinance
  if (Finance[EVT_1.row][EVT_1.column].NegativeRange = "F")
  then
    {Finance[EVT_1.row][EVT_1.column].WorstValue ::= min(toFloat(EVT_1.slotvalue));
      Finance[EVT_1.row][EVT_1.column].BestValue ::= max(toFloat(EVT_1.slotvalue));
    }
  discard(EVT_1);
}
```

This rule is triggered by the alarm event that occurs when the current value slot is updated and the range of worst to best value is positive. The negative range is false, which is positive.

```java
if (new current value is less than the worst value, set the worst value to be equivalent to the current value, so that the out-of-range values do not spoil the calculations. Values out of the worst and best values need not necessarily be between 1 and 10. This spoils the subsequent scores based on that erroneous calculation.

Finance[EVT_1.row][EVT_1.column].BestValue ::= max(toFloat(EVT_1.slotvalue));
```

If the new current value is greater than the best value, set the best value to be equivalent to the current value.

```java
rule Finance_CurrentValue_LT activated by EVT_1 of BPEVENT_INFOPAD_ALARM::currentcheckFinance
  if (Finance[EVT_1.row][EVT_1.column].NegativeRange = "T")
  then
    {Finance[EVT_1.row][EVT_1.column].WorstValue ::= max(toFloat(EVT_1.slotvalue));
      Finance[EVT_1.row][EVT_1.column].BestValue ::= min(toFloat(EVT_1.slotvalue));
    }
  discard(EVT_1);
}
```

This instruction generates an event of the exact same type with the value equal to UpdateFinanceScore, which triggers a rule to update the finance score in this cell.

```java
rule Finance_CurrentValue_update activated by EVT_1 of BPEVENT_INFOPAD_ALARM::updateFinanceScore
  if (Finance[EVT_1.row][EVT_1.column].NegativeRange = "T")
  then
    {Finance[EVT_1.row][EVT_1.column].WorstValue ::= max(toFloat(EVT_1.slotvalue));
      Finance[EVT_1.row][EVT_1.column].BestValue ::= min(toFloat(EVT_1.slotvalue));
    }
  discard(EVT_1);
}
```

This instruction generates an event of the exact same type with the value equal to UpdateFinanceScore, which triggers a rule to update the finance score in this cell.
This event updates the finance score of the cell which originated the event and triggers this event only when it is generated from an alarm condition.

```
then
{Finance[EVT_1.row][EVT_1.column].Score :=
 (((Finance[EVT_1.row][EVT_1.column].CurrentValue -
Finance[EVT_1.row][EVT_1.column].WorstValue)/
(Finance[EVT_1.row][EVT_1.column].BestValue -
Finance[EVT_1.row][EVT_1.column].WorstValue))*10)/
Finance[EVT_1.row]["FinanceScore"].RelativeWeight);

discard(EVT_1);
}
```

The current, worst, and best value sub-equation gives the percentage that the current value is between worst value and best value between 0 and 1. Multiplying by 10 gives you the range of the score from 0 to 10. Dividing by the sum of the relative weights normalizes the score when the sum is greater than one. If the score equals one, then it is inconsequential. This prevents user error from breaking the calculations.

```
rule Finance_Weight activated by EVT_1 of
BPEVENT_INFOPAD_ALARM::relativecheckFinance
```

This event updates the overall relative weight when one of the individual KPI relative weights change. This also tends to renormalize all of the scores in the infopad because they depend on the overall relative weight which changed.

```
if (EVT_1.column != "7")
```

It is not equal to the overall relative weight slot from the last column which is FinanceScore or 7.

```
then

{Finance[EVT_1.row]["FinanceScore"].RelativeWeight :=
(Finance[EVT_1.row][1].RelativeWeight +
Finance[EVT_1.row][2].RelativeWeight +
Finance[EVT_1.row][3].RelativeWeight +
Finance[EVT_1.row][4].RelativeWeight +
Finance[EVT_1.row][5].RelativeWeight +
Finance[EVT_1.row][6].RelativeWeight);

This illustrates the overall relative weight slot is the sum of the other 6 relative weights reflected here in the update call.
```

```
generate(type: "BPEVENT_INFOPAD_ALARM",value :
"updateFinanceScore",slot : "4",row : EVT_1.row, column : 1, table :
"", NegativeRange : "");
generate(type: "BPEVENT_INFOPAD_ALARM",value :
"updateFinanceScore",slot : "4",row : EVT_1.row, column : 2, table :
"", NegativeRange : "");
generate(type: "BPEVENT_INFOPAD_ALARM",value :
"updateFinanceScore",slot : "4",row : EVT_1.row, column : 3, table :
"", NegativeRange : "");
generate(type: "BPEVENT_INFOPAD_ALARM",value :
"updateFinanceScore",slot : "4",row : EVT_1.row, column : 4, table :
"", NegativeRange : "");
generate(type: "BPEVENT_INFOPAD_ALARM",value :
"updateFinanceScore",slot : "4",row : EVT_1.row, column : 5, table :
"", NegativeRange : "");
generate(type: "BPEVENT_INFOPAD_ALARM",value :
"updateFinanceScore",slot : "4",row : EVT_1.row, column : 6, table :
"", NegativeRange : "");

discard(EVT_1);
```
You can renormalize all of the scores as the overall relative weight has changed.

```
rule Finance_Score activated by EVT_1 of
BPEVENT_INFOPAD_ALARM::scorecheckFinance

This rule updates the perspective scores when one of the individual KPI scores has changed.

if (EVT_1.column != "7")

...and it was not the overall perspective score in column FinanceScore or seven.

then
    {Finance[EVT_1.row]["FinanceScore"].Score :=
    ((Finance[EVT_1.row][1].Score*Finance[EVT_1.row][1].RelativeWeight) +
    (Finance[EVT_1.row][2].Score*Finance[EVT_1.row][2].RelativeWeight) +
    (Finance[EVT_1.row][3].Score*Finance[EVT_1.row][3].RelativeWeight) +
    (Finance[EVT_1.row][4].Score*Finance[EVT_1.row][4].RelativeWeight) +
    (Finance[EVT_1.row][5].Score*Finance[EVT_1.row][5].RelativeWeight) +
    (Finance[EVT_1.row][6].Score*Finance[EVT_1.row][6].RelativeWeight));
    discard(EVT_1);
}
```

The overall score is determined by adding the score of each KPI multiplied by its relative weight.

```
rule Finance_PScore activated by EVT_1 of
BPEVENT_INFOPAD_ALARM::scorecheckFinance

This rule updates the perspective scores when one of the individual KPI scores has changed.

if (EVT_1.column != "7")

...and it was not the overall perspective score in column FinanceScore or seven.

then
    {generate(type : "UpdateBalancedScorecard", value :
    "BalancedScorecardscoreupdate", row : EVT_1.row);
    discard(EVT_1);
}
```

The scorecard is updated.

```
rule BP ServerUpdateFinance_0_0 activated by EVT_1 of
BP Server::W_COMPLETED
if (EVT_1.PROCESSTEMPLATENAME = "BalancedScorecard") and
(EVT_1.WORKSTEPNAME = "FinanceSC")
then
    {Finance[common::common_rules::MonthLookupInfopad[EVT_1.date.month]
[1].monthName+toString(EVT_1.date.year)][1].CurrentValue +=
    toFloat(EVT_1.FinancePR);
    discard(EVT_1);
}
```

```
rule BP ServerUpdateFinance_1_0 activated by EVT_1 of
BP Server::W_COMPLETED
if (EVT_1.PROCESSTEMPLATENAME = "BalancedScorecard") and
(EVT_1.WORKSTEPNAME = "FinanceSC")
then
    {Finance[common::common_rules::MonthLookupInfopad[EVT_1.date.month]
[1].nonthName+toString(EVT_1.date.year)][2].CurrentValue ::=
    avg(toFloat(EVT_1.FinanceSR),Finance[common::common_rules::
MonthLookupInfopad[EVT_1.date.month][1].nonthName+toString(EVT_1.
date.year)][2].count++,1);
    discard(EVT_1);
}
```

```
rule BP ServerUpdateFinance_2_0 activated by EVT_1 of
BP Server::W_COMPLETED
if (EVT_1.PROCESSTEMPLATENAME = "BalancedScorecard") and
(EVT_1.WORKSTEPNAME = "FinanceSC")
then
    {Finance[common::common_rules::MonthLookupInfopad[EVT_1.date.month]
[1].nonthName+toString(EVT_1.date.year)][2].count++;
    discard(EVT_1);
}
```

```
rule BP ServerUpdateFinance_3_0 activated by EVT_1 of
BP Server::W_COMPLETED
if (EVT_1.PROCESSTEMPLATENAME = "BalancedScorecard") and
(EVT_1.WORKSTEPNAME = "FinanceSC")
then
    {Finance[common::common_rules::MonthLookupInfopad[EVT_1.date.month]
[1].nonthName+toString(EVT_1.date.year)][3].CurrentValue +=
    toFloat(EVT_1.FinanceSR);
    discard(EVT_1);
}
```

```
rule BP ServerUpdateFinance_4_0 activated by EVT_1 of
BP Server::W_COMPLETED
if (EVT_1.PROCESSTEMPLATENAME = "BalancedScorecard") and
(EVT_1.WORKSTEPNAME = "FinanceSC")
then
    {Finance[common::common_rules::MonthLookupInfopad[EVT_1.date.month]
[1].nonthName+toString(EVT_1.date.year)][3].CurrentValue ::=
    avg(toFloat(EVT_1.FinanceSR),Finance[common::common_rules::
MonthLookupInfopad[EVT_1.date.month][1].nonthName+toString(EVT_1.
date.year)][3].count++,1);
    discard(EVT_1);
}
```
Updates the CurrentValue of the first KPI by the value in the dataslot Finance1. This is a dynamic row selection which takes the name of the current month plus the current year to designate the current row. The MonthLookupInfopad holds the names of the months, and the month number looks up the correct name before the year is appended to it.

rule BalancedScorecardimp_BalancedScorecard activated by EVT_1 of BPEVENT_INFOPAD_ALARM::updateBalancedScorecardScores

This rule updates the overall balanced score when any of the perspective weights in the BalancedScorecardimp infopad are updated. Trigger this rule when the event comes from an alarm condition.

then
{
    generate(type : "UpdateBalancedScorecard", value : 
        "BalancedScorecardscoreupdate", row : 
            BalancedScorecard.rowCount());
    discard(EVT_1);
}

This action generates an update BalancedScorecard event which updates the overall score.

rule BalancedScorecard_update activated by EVT_1 of UpdateBalancedScorecard::BalancedScorecardscoreupdate

Trigger this rule when the event type is UpdateBalancedScorecard and the event value is BalancedScorecardscoreupdate, that is either a perspective score or perspective weight was updated, thus necessitating an update to the overall balanced score.

then
{
    BalancedScorecard[EVT_1.row][1].Score := 
        ((BalancedScorecardimp["Finance"] ["Current "].Weight*Finance[EVT
_1.row]["FinanceScore"]).Score)+(BalancedScorecardimp["Customer 
"] ["Current "].Weight*Customer[EVT_1.row]["CustomerScore"]).Score
}
The overall score is the sum of each perspective score multiplied by the perspective weight. This score is normalized by dividing by the sum of the perspective weights.

```plaintext
rule addRows activated by EVT_1 of
  AddBalancedScorecardScorecardRows::monthly
```

This rule is triggered by a scheduled event with type AddScorecardRows and value monthly. The first time this rule runs, it is triggered by the schedule action in the initialization section. Subsequent triggers are from events scheduled at the end of this rule. The sample implementation updates the rows on a monthly basis.

```plaintext
then
  var addedRowIndex = Finance.add();

This action adds a row to the dynamic infopad FinanceInfopad. The index of the newly created row is stored in addedRowIndex. The following explanations apply to the FinanceInfopad only. The actions for the other perspective infopads are similar.

```plaintext
if(EVT_1.date.month != 12)
  Finance.setRowLabel(addedRowIndex,
    common::common_rules::MonthLookupInfopad[EVT_1.date.month+1][1].monthName+EVT_1.date.year);
else
  Finance.setRowLabel(addedRowIndex,
    common::common_rules::MonthLookupInfopad[1][1].monthName+(EVT_1
    .date.year+1));
```

This action sets the row label for the newly created row. The row label is of the form <monthname><year>, for example, March2005. The infopad MonthLookupInfopad contains the string names of the months. Also, as this rule is run on the last day of the month, you can lookup the name of the current month + 1, with december(12) being a special case. The addedRowIndex value is the row number to add the new label to.

```plaintext
Finance[addedRowIndex][1].count :=
Finance[addedRowIndex-1][1].count;
Finance[addedRowIndex][1].WorstValue :=
Finance[addedRowIndex-1][1].WorstValue;
Finance[addedRowIndex][1].BestValue :=
Finance[addedRowIndex-1][1].BestValue;
Finance[addedRowIndex][1].CurrentValue :=
Finance[addedRowIndex-1][1].CurrentValue;
Finance[addedRowIndex][1].RelativeWeight :=
Finance[addedRowIndex-1][1].RelativeWeight;
Finance[addedRowIndex][1].Score := Finance[addedRowIndex-1][1].Score;
Finance[addedRowIndex][1].Description :=
Finance[addedRowIndex-1][1].Description;
Finance[addedRowIndex][1].Unit := Finance[addedRowIndex-1][1].Unit;
Finance[addedRowIndex][1].NegativeRange :=
Finance[addedRowIndex-1][1].NegativeRange;
```
The above action initializes the values in the new row by assigning them the current values of the previous one. This is done for all the columns in the new row.

```plaintext
Finance[addedRowIndex]["FinanceScore"].Score :=
Finance[addedRowIndex-1]["FinanceScore"].Score;
Finance[addedRowIndex]["FinanceScore"].RelativeWeight :=
Finance[addedRowIndex-1]["FinanceScore"].RelativeWeight;
```

The last column must contain only the sum of the relative weights and scores, for this perspective.

```plaintext
addedRowIndex := BalancedScorecard.add();
if(EVT_1.date.month != 12)
   BalancedScorecard.setRowLabel(addedRowIndex,
      common::common_rules::MonthLookupInfopad[EVT_1.date.month+1][1].nonthName+EVT_1.date.year);
else
   BalancedScorecard.setRowLabel(addedRowIndex,
      common::common_rules::MonthLookupInfopad[1][1].nonthName+(EVT_1.date.year+1));
BalancedScorecard[addedRowIndex][1].Score :=
BalancedScorecard[addedRowIndex-1][1].Score;
```

The above action performs similar actions for the Main BalancedScorecard infopad as for FinanceInfoPad and the other perspective infopads.

```plaintext
schedule("BalancedScorecardRowCreator", (((YEAR:((NOW).year())/
      MONTH:((NOW).month()))+(1*MONTH))-(1*DAY)),type:"AddBalancedScorecardScorecardRows",value:"monthly");
discard(EVT_1);
```

The above action schedules the event which creates the rows for the next month.

```plaintext
finalize
(discard(BalancedScorecardImp);
discard(Finance);
discard(Customer);
discard(Internal);
discard(Innovation);
discard(BalancedScorecard);
}
```

This rule is run when you unload the rules from BPM Events. It cleans up the infopads from memory and the database, and is extremely useful during debugging.
Glossary

**ACL manager**
In Business Process Server, Access Control List Manager provides a finer, more precise control over user access rights for resources and actions.

**Activity workstep**
In Business Process, the basic unit of work; must be performed by one or more human performers (valid individual user, multiple users or user group).

**Adapter**
A Java class that integrates remote, third party classes and actions with Business Process. An adapter can automate certain functions and tasks performed by a remote server or other external systems.

**Administration**
A module in Business Process Portal enabling the administrator to perform tasks such as installing/uninstalling applications, modifying configuration parameters controlling Business Process operations, and manage users, groups and access control. The Administration module is visible only to application users who have permissions to access it.

**Application**
In Business Process, an application is an installed, executable business process that automates a business flow.

**Balanced scorecard**
A management application in the Management module that measures performance by analyzing how an organization’s business activities help it achieve its strategic goals. The Balanced Scorecard provides an analysis from a range of perspectives.

**BAM**
Business Activity Management combines Business process management with strategic and analytical information on specific business performance indicators, providing real-time status information and identifying critical events to assist senior management in making informed business decisions.
BPM Events
A Business Process Server component that provides an open event-driven rule engine to formulate and enforce policies in business applications.

BPM Webflow
A Business Process Server component that enables users to develop customizable, sophisticated presentation flows for business processes, install them as Web applications, and execute them on their Web browsers.

BPEL
BPEL (Business Process Execution Language) for Web services is an XML-based language designed to enable task-sharing for a distributed computing or grid computing environment - including across multiple organizations - using a combination of Web services.

BPMN
BPMN (Business Process Modelling Notation) provides businesses with the capability of defining and understanding their internal and external business procedures through a Business Process Diagram giving organizations the ability to communicate these procedures in a standard manner.

BP Server
A Business Process Server component that provides a flexible, lightweight, scalable workflow process engine for intranets, extranets, and the Internet.

Business calendar
A Business Process Server feature that accurately calculates the Due Date of tasks, and provides support for multiple business calendars across different time zones.

Business flow
The logical sequence of process activities, related to one another by a triggering activity, to achieve an outcome. It represents a business process that begins with a commitment and ends with the termination of that commitment. In Business Process Server, business flow includes Workflow (the flow of all human-performed activities), integration flow (the flow of activities performed by systems) and presentation flow (from a user’s viewpoint, the flow of data from one Web page to the next).

Business logic
The control flow and information flow among worksteps that define a business process.

Business object
A representation of an activity in the business domain, including its name, definition, attributes, behavior, relationships and constraints.

Business process
A process involving multiple worksteps in the form of operations, interactions and notifications performed by a user, group of users, an external adapter, or a script.

Business Process Server application
An application is an implementation of a business process. It can contain one or more process templates, performers, adapters, customized forms or rules. An application can be published, installed and run on BP Servers. In Business Process Server, an application is an installed, executable business process that automates a Workflow.
**Business Process Server Web services**

A Business Process Server component that allows application developers to; a) publish their applications as Web services, and b) find and convert other available Web services on the Internet into Business Process Server applications.

**Business Process Portal**

A Business Process Server component that offers users, managers, administrators and developers a unified, customizable portal for single sign-on access to all Business Process Server functionalities to which they are granted permission.

**Business process management**

The concept of guiding work activities through a multi-step business process in order to improve performance and reduce costs within and across functional business units.

**Business Process Modeler**

A stand-alone component that enables users to design templates for basic business processes.

**Business rule**

A combination of elements, including validation edits, logon verifications, database lookups, policies and transformations, that represent an enterprise’s way of doing business.

**Control flow**

The sequences of worksteps and workstep conditions, as defined in a process template in Progress Developer Studio for OpenEdge or Business Process Modeler.

**Dashboard**

A Business Process Server feature that provides a graphic overview of the status of several business processes on a single Web page, enabling users to monitor the progress of each process. Users can view business processes across all applications or for a selected application.

**Dataslot**

A data placeholder that persists through the entire process and defines the information flow of the business process. Dataslots are associated with processes, where they can add information into (Input type) or out of (Output type) worksteps, and appear as editable or read-only fields on a user’s interface.

**Expression editor**

A Business Process Server tool that enables users to define complex conditional expressions within a Decision gateway to support their business requirements.

**Group**

In Business Process Server, an entity that has as members valid users or other groups who perform related work and have authorized access to specific components.

**Heatmap**

A Business Process Server feature that provides a convenient, graphical tool for managers to visually locate the bottlenecks in the process execution. It helps managers to get an overview of the status of the currently active instances, identify suspended instances, and analyze the history of the completed instances.

**Home**

A module in Business Process Portal through which users interact with Business Process Server. Using the Home module, users complete entries to various tasks and applications, update profile, set preferences, and link to the support infrastructure required to achieve these tasks. The Home module is the primary interface for application users.
Infopad
In Business Process Server, a data structure used to capture business metrics, typically displayed as a table with one or two dimensions.

Instance
An individual object within a specific class. In Business Process Server, a self-contained unit that is created each time you use a process template to run a Business Process Server application.

KPI
Key Performance Indicator, used in the Balanced Scorecard system, that provides the data translating enterprise goals into a set of measurable objectives.

Managed Adapter
In Business Process Server, a Managed Adapter is an implementation of an adapter interface that facilitates data exchange between Business Process Server processes and external applications.

Management
A module in Business Process Portal enabling the managers to query, report, and control processes and resources for application users. The Management module is visible only to application users who have permissions to access it.

Migration
The process of moving from the use of one operating environment to another operating environment that is typically seen as improvement. Migration can involve moving to new hardware, new software, or both. It may involve a new application, another type of database, or a redesigned network. Migration is also used to refer simply to the process of moving data from one storage device to another. Business Process Server supports data migration as well as application migration.

Performer
An entity that executes a workstep. Depending on the workstep type, the performer can be a human user, a group of users, an adapter or other external performer, or a script.

Presentation flow
The flow of information and user input from one interface to the next. Typically related to a single Activity workstep in the process and generated in a BPM Webflow environment.

Process engine
Orchestrates the execution of business processes and also coordinates conversations among process engines based on public processes, which forms the backbone of global business collaboration.

Process refresh
A Business Process Server feature for replacing the installed process without versioning, facilitating the running process instances to refresh and seamlessly adapt to the new Workflow.

Process template
In Business Process Server, a model of business flow that includes worksteps, connectors and dataslots. After users publish and install it as an application in Business Process Server folder structure, they can use the application to create process instances.

Progress Developer Studio for OpenEdge
An Integrated Development Environment for Business Process Server that enables application users to develop and publish a Business Process Server application without leaving the development environment.
Role
The actions and activities assigned to a valid application user who is a member of a group. In Business Process Server, only members of a group can be assigned a role. A role indicates the relationships of the user in a group context.

Rollback
In Business Process Server, a feature that restarts the Workflow from a workstep previously selected as the rollback point in the process, performed automatically in the event of a failure.

Rule wizard
An interactive utility that enables application users to quickly develop rules that can be applied to a business process.

Swim lanes
Used in Workflow diagrams to organize complex processes across functional boundaries. For example, seen as horizontal lines on a process map, swim lanes can be used to place individual task steps into different categories that depend on task ownership.

Task
In Business Process Server, a performer is assigned one or more work items that the performer sees as tasks. There are two types of tasks: Assigned, which are assigned specifically to you; and Available, which are available to be performed by you or other members of your user group.

User
In Business Process Server, a valid human performer with authorized access to specific modules.

Workflow
The logical sequence of activities performed by human performers. Workflow includes the tasks, procedural steps, organizations or people involved, required input and output information, and tools needed for each activity in a business process.