Copyright

© 2017 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.

These materials and all Progress® software products are copyrighted and all rights are reserved by Progress Software Corporation. The information in these materials is subject to change without notice, and Progress Software Corporation assumes no responsibility for any errors that may appear therein. The references in these materials to specific platforms supported are subject to change.

Business Making Progress, Corticon, DataDirect (and design), DataDirect Cloud, DataDirect Connect, DataDirect Connect64, DataDirect XML Converters, DataDirect XQuery, Deliver More Than Expected, Icenium, Kendo UI, Making Software Work Together, NativeScript, OpenEdge, Powered by Progress, Progress, Progress Software Developers Network, Rollbase, SequeLink, Sitefinity (and Design), SpeedScript, Stylus Studio, TeamPulse, Telerik, Telerik (and Design), Test Studio, and WebSpeed are registered trademarks of Progress Software Corporation or one of its affiliates or subsidiaries in the U.S. and/or other countries. AccelEvent, Analytics360, AppsAlive, AppServer, Arcade, BravePoint, BusinessEdge, DataDirect Spy, DataDirect SupportLink, DevCraft, DigitalFactory, Fiddler, Future Proof, High Performance Integration, JustCode, JustDecompile, JustMock, JustTrace, OpenAccess, ProDataSet, Progress Arcade, Progress Profiles, Progress Results, Progress RFID, Progress Software, ProVision, PSE Pro, SectorAlliance, Sitefinity, SmartBrowser, SmartComponent, SmartDataBrowser, SmartDataObjects, SmartDataView, SmartDialog, SmartFolder, SmartFrame, SmartObjects, SmartPanel, SmartQuery, SmartViewer, SmartWindow, WebClient, and Who Makes Progress are trademarks or service marks of Progress Software Corporation and/or its subsidiaries or affiliates in the U.S. and other countries. Java is a registered trademark of Oracle and/or its affiliates. Any other marks contained herein may be trademarks of their respective owners.

Please refer to the Release Notes applicable to the particular Progress product release for any third-party acknowledgements required to be provided in the documentation associated with the Progress product.

The Release Notes can be found in the OpenEdge installation directory and online at: https://community.progress.com/technicalusers/w/openedgegeneral/1329.openedge-product-documentation-overview.aspx.

For the latest documentation updates see OpenEdge Product Documentation on Progress Communities: (https://community.progress.com/technicalusers/w/openedgegeneral/1329.openedge-product-documentation-overview.aspx).

December 2014

Last updated with new content: Release 11.5.0

Updated: 2017/03/24
# Table of Contents

**Preface** ...............................................................................................................15  
  - About this documentation ..................................................................................15  
  - User types ..........................................................................................................15  
  - Information on documentation ..........................................................................16  
  - Conventions used in this manual .......................................................................17  
  - Product support contact information ................................................................18  

**Chapter 1: About Business Process Server** ..................................................21  

**Chapter 2: Business Process Server in OpenEdge** .......................................23  
  - BP Server features ..............................................................................................23  
  - Duplicating a mapped name in an external workstep .........................................26  
  - BP Server MDB control ....................................................................................27  
    - Setting the configuration parameters ...........................................................27  
  - Archiver ............................................................................................................28  

**Chapter 3: Business calendar** ......................................................................29  
  - Business calendar tags .......................................................................................30  
    - Working time tags ..........................................................................................30  
    - Non-working time tags ..................................................................................30  
    - Time zone and locale tags ..............................................................................33  
  - Multiple calendars ............................................................................................33  
    - Precedence for calendars ..............................................................................34  
  - Calculating the due date ....................................................................................34  
    - Workstep instance due date ...........................................................................35  
    - Pre-script in workstep due date ....................................................................35  
    - Work item due date .......................................................................................36  
  - Calculating the duration ....................................................................................36  
    - Working hours for regular business hours .....................................................37  
    - Working hours for special business hours .....................................................37  
    - Working hours for half-day holiday ................................................................38  
  - BP Server API for due date calculation ..............................................................39  
  - BizCalendar SVO ..............................................................................................39  

**Chapter 4: Process refresh** ..........................................................................41  
  - Changes during refresh ....................................................................................41
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naming process templates</td>
<td>66</td>
</tr>
<tr>
<td>Versioning and APIs</td>
<td>66</td>
</tr>
<tr>
<td>Versioning example</td>
<td>67</td>
</tr>
<tr>
<td>Creating process templates</td>
<td>68</td>
</tr>
<tr>
<td>Installing process template</td>
<td>68</td>
</tr>
<tr>
<td>Creating versions</td>
<td>68</td>
</tr>
<tr>
<td>Accessing versions</td>
<td>68</td>
</tr>
<tr>
<td>Getting version list</td>
<td>68</td>
</tr>
<tr>
<td>Getting version ID</td>
<td>69</td>
</tr>
<tr>
<td>Getting version state</td>
<td>69</td>
</tr>
<tr>
<td>Chapter 10: Workstep performers</td>
<td>71</td>
</tr>
<tr>
<td>Types of performers</td>
<td>71</td>
</tr>
<tr>
<td>Specifying performer precedence</td>
<td>72</td>
</tr>
<tr>
<td>Understanding workstep performer hierarchy</td>
<td>72</td>
</tr>
<tr>
<td>Default performer for tasks assigned to user groups</td>
<td>73</td>
</tr>
<tr>
<td>Excluding performers for worksteps</td>
<td>73</td>
</tr>
<tr>
<td>Specifying the exclude performers</td>
<td>74</td>
</tr>
<tr>
<td>Actual performer of completed work item</td>
<td>74</td>
</tr>
<tr>
<td>Workstep properties in OpenEdge</td>
<td>75</td>
</tr>
<tr>
<td>Chapter 11: Dynamic user profile</td>
<td>77</td>
</tr>
<tr>
<td>Working with the UserProfile object</td>
<td>77</td>
</tr>
<tr>
<td>Using the UserProfile object with queues</td>
<td>78</td>
</tr>
<tr>
<td>Chapter 12: Query service</td>
<td>79</td>
</tr>
<tr>
<td>Working with queries</td>
<td>79</td>
</tr>
<tr>
<td>QSWorkItem</td>
<td>81</td>
</tr>
<tr>
<td>QSWorkStepInstance</td>
<td>81</td>
</tr>
<tr>
<td>QSProcessInstance</td>
<td>81</td>
</tr>
<tr>
<td>Working with filters</td>
<td>81</td>
</tr>
<tr>
<td>Filter properties</td>
<td>82</td>
</tr>
<tr>
<td>Constructing Filters</td>
<td>83</td>
</tr>
<tr>
<td>Filter constraints</td>
<td>84</td>
</tr>
<tr>
<td>Parameterized condition support for filter</td>
<td>84</td>
</tr>
<tr>
<td>Query service results</td>
<td>84</td>
</tr>
<tr>
<td>Configuring Business Process Server for query service</td>
<td>86</td>
</tr>
<tr>
<td>Chapter 13: Smart value objects</td>
<td>87</td>
</tr>
<tr>
<td>Process SVOs</td>
<td>88</td>
</tr>
<tr>
<td>Collection SVOs</td>
<td>89</td>
</tr>
<tr>
<td>Dataslot SVOs</td>
<td>89</td>
</tr>
</tbody>
</table>
Chapter 14: Application management.................................................................91
  Accessing applications......................................................................................91
  Using BP Server APIs......................................................................................92
  Creating applications......................................................................................92
  Retrieving applications...................................................................................93
  Removing applications....................................................................................93

Chapter 15: Process template management.....................................................95
  Accessing process templates...........................................................................96
  Using BP Server APIs for process template management..............................96
  Creating process templates............................................................................96
  Retrieving process templates.........................................................................97
  Working with process template states............................................................97
    Getting the process template states...............................................................98
    Changing the process template states..........................................................99
  Setting the process template priority.............................................................99
  Managing the process template information...............................................100
    Setting process template attributes.............................................................100
    Retrieving process template information..................................................100
    Getting dataslot information.......................................................................100
  Working with dynamic process templates...................................................100
  Working with subprocesses..........................................................................100
    Subprocess alias.........................................................................................101

Chapter 16: Process instance management......................................................103
  Using BP Server APIs for process instance management..............................104
  Creating process instances............................................................................104
  Retrieving process instances.......................................................................104
    Retrieving process instances from BP Server.............................................104
    Retrieving process instances from class ProcessTemplate..........................105
  Working with process instance states...........................................................105
    Getting the process instance state...............................................................106
    Changing the process instance state..........................................................107
  Setting the process instance priority.............................................................107
  Removing process instances.........................................................................108
  Working with dynamic process instances...................................................108

Chapter 17: Workstep template management..................................................111
  APIs for workstep templates.........................................................................111
  Retrieving workstep templates....................................................................112
  Working with workstep template states.......................................................113
Chapter 18: Work item management.................................................................121

Accessing work items.........................................................................................................122
APIs for work items..................................................................................................................122
Working with work item states................................................................................................122
Working with work item priority..............................................................................................123
Assigning work items............................................................................................................124
Assigning an available work item..........................................................................................124
Reassigning an assigned work item.........................................................................................124
Reassigning bulk work items.................................................................................................124
Making an assigned work item available..............................................................................125
Using dataslot information.....................................................................................................125
Getting dataslot information..................................................................................................125
Getting metadata information...............................................................................................126
Completing work items..........................................................................................................127
Accessing performing apps.....................................................................................................127
APIs for performing apps.......................................................................................................127
Working with work item performing apps..........................................................................128
Getting work items from other classes................................................................................128
Getting work items from BP Server.......................................................................................128
Getting work items from class ProcessInstance.................................................................129
Getting work items from class WorkStepInstance.............................................................129

Chapter 19: Workstep Instance Management.........................................................131

Accessing workstep instances.............................................................................................132
APIs for workstep instances.................................................................................................132
Retrieving workstep instances..............................................................................................132
Working with workstep instance types................................................................................132
Working with the workstep instance states.........................................................................132
Working with the workstep instance priority.......................................................................133
Chapter 20: Dynamic work item management using queues.............135
Pass and pull mechanisms for work items .................................................................135
Work item pull............................................................................................................136
Work item management................................................................................................136
Retrieving available work items..................................................................................136
Specifying timer action for available work items......................................................137
Completing assigned work items...............................................................................137
Removing user / group from a queue.........................................................................138
Removing a queue.......................................................................................................138

Chapter 21: Dataslot management.................................................................139
Using BP Server APIs for dataslot management......................................................141
Using dataslots...........................................................................................................141
Retrieving dataslots ..................................................................................................141
Getting dataslot information......................................................................................142
Using DataSlotTemplate............................................................................................142
Retrieving DataSlotTemplates..................................................................................142
Setting attributes of DataSlotTemplate.......................................................................142
Saving attributes of DataSlotTemplate.......................................................................142
Working with different types of dataslots.................................................................143
Document dataslots..................................................................................................143
INTEGER dataslots....................................................................................................147
DATETIME-TZ dataslots...........................................................................................148
OBJECT dataslots.....................................................................................................149
LIST dataslots..........................................................................................................150
CHARACTER dataslots.............................................................................................150
MAP dataslots..........................................................................................................150

Chapter 22: Configuring performers.........................................................151
Remote JNDI external performer application..........................................................151
Configuring remote JNDI properties in oebpsjndi.properties file.........................151
Adding a remote external performer workstep to the process template..................154
Creating and deploying the ear for the remote external performer..........................154
Local JNDI external performer application..............................................................154
Adding a local adapter workstep to the process template.........................................154
Mapping the JNDI name of the external performer to a fully qualified class name.......154
Callback code example.............................................................................................155
EJB performer in atomic workstep.............................................................................155
Chapter 23: E-mail templates for task completion.................................159

Enabling task completion by e-mail.......................................................160
Variables..............................................................................................160
   Formal reference notation...............................................................160
Expressions........................................................................................161
Variables using formal notation..........................................................162
System variables..................................................................................162
   BP Server variables..........................................................................162
   E-mail variables................................................................................163
User defined dataslot values...............................................................164
Pre-populated output dataslot values..................................................164
API for sending e-mails to assigned task............................................165
E-mail recipients................................................................................165
E-mail subject.....................................................................................165
E-mail backup.....................................................................................166
E-mail search.....................................................................................166
E-mail clients.....................................................................................166
E-mail security...................................................................................167
Types of e-mail templates...................................................................167
Text e-mail template..........................................................................168
   E-mail templates locations and file format.....................................168
   BP Server default e-mail templates...............................................169
   Custom text e-mail template..........................................................170
   E-mail template viewer...................................................................170
HTML e-mail templates.....................................................................170
   SendHtmlEmail method.................................................................171
   HTML images...............................................................................171
   Multipart e-mail content..............................................................172
   HTML template with dynamic content..........................................173
   Default HTML template..............................................................174
   Custom HTML e-mail template.....................................................174
   HTML composer............................................................................176
   Guidelines for custom template....................................................177
   Reply e-mail..................................................................................178
   Decoding of document file names................................................178
   Features of default template.........................................................178
   Dataslot handling..........................................................................179
   Text E-mail Templates..................................................................179
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>
</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>User type</th>
<th>Responsibilities</th>
</tr>
</thead>
<tbody>
<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>

<table>
<thead>
<tr>
<th>If you are the …</th>
<th>Read the …</th>
</tr>
</thead>
</table>
| Application User | *Business Process Portal User’s Guide*  
                   *First Steps Guide*  
                   *Terminology Guide* |
| Manager          | *Business Process Portal Manager’s Guide*  
                   *Business Process Portal User’s Guide*  
                   *Terminology Guide* |
### 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>file path</td>
<td>Indicates folder paths and file names.</td>
</tr>
</tbody>
</table>

For the latest Business Process Server documentation updates, see OpenEdge Product Documentation on PSDN (http://communities.progress.com/pcom/docs/DOC-16074).
### Convention (styles and terms)

<table>
<thead>
<tr>
<th>Style</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>italic</em></td>
<td>Indicates book titles.</td>
</tr>
<tr>
<td>monospace</td>
<td>Represents code segments or examples.</td>
</tr>
<tr>
<td>backward slash</td>
<td>Indicates the path in Windows environment.</td>
</tr>
<tr>
<td>&quot;&quot;</td>
<td>For UNIX environment, replace with forward slash &quot;/&quot;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Variable</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>OEBPS_HOME or</td>
<td>C:\Progress\OpenEdge\oebpm\server</td>
</tr>
<tr>
<td>%OEBPS_HOME%</td>
<td></td>
</tr>
<tr>
<td>STUDIO_HOME or</td>
<td>C:\Progress\OpenEdge\oebpm\studio</td>
</tr>
<tr>
<td>%STUDIO_HOME%</td>
<td></td>
</tr>
<tr>
<td>JBOSS_HOME or</td>
<td>C:\Progress\OpenEdge\oebpm\jboss</td>
</tr>
<tr>
<td>%JBOSS_HOME%</td>
<td></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>
<tbody>
<tr>
<td>Web site</td>
<td><a href="http://progresslink.progress.com/supportlink">http://progresslink.progress.com/supportlink</a></td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>Telephone¹</td>
<td>1-781-280-4999 for US, Latin America and Canada</td>
</tr>
<tr>
<td></td>
<td>1-781-280-4543 for the Product Support Fax Line</td>
</tr>
<tr>
<td>Postal Address¹</td>
<td>Progress Software Corporation</td>
</tr>
<tr>
<td></td>
<td>14 Oak Park Drive</td>
</tr>
<tr>
<td></td>
<td>Bedford, MA 01730, USA.</td>
</tr>
</tbody>
</table>

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.
About Business Process Server

Business Process Server (BP Server) is a flexible, lightweight, scalable workflow process engine for intranets, extranets, and the Internet. BP Server generates events when it creates workflow process instances, completes worksteps, and completes workflow instances. BP Server also generates events when a process template is created or installed, or a BP Server session is started or ended. Each BP Server process uses JDBC to connect to database server processes and store events in the database. These events contain the complete application state transition information and also serve as an audit trail. BP Server stores these events as rows in the database table BizEvent. The BizEvent table captures all the information encoded in an event object, and includes an eventID that uniquely identifies the event within an Business Process Server installation. BPM Process Store and BPM Events rely on the eventID to sequentially process all events.

End users access applications, obtain information to perform their tasks, and launch a new process from the BP Server through Business Process Portal.

BPM Process Store is the repository feature of BP Server. The BPM Process Store daemon runs as a batch processing process. It wakes up periodically according to a configurable sleep time parameter, selects unprocessed events from the BPM Events table, processes them, and goes back to sleep if there are no more events to process. BPM Process Store uses a configuration parameter called batchSize to determine how many events to fetch from database in each JDBC call. Depending on the event type, the BPM Process Store server issues database commands to insert or update data in the Process Instance, Workstep and Workitem tables. For example, on receiving a "PI_CREATED" event, BPM Process Store inserts a row into the table "ProcessInstance" containing all the initial states of the newly created process instance.

BPM Events is the execution engine for rule sets and it operates in conjunction with the BP Server. The BPM Events server also wakes up periodically to process events in the Event table. BPM Events reads the eventSource.eventBatchSize configuration parameter to determine its batch size. However, instead of populating process tables, BPM Events evaluates a set of user-defined rules against each event to see if an action should be triggered. For example, you might specify a rule to instruct BPM Events to update a slot value of an infopad cell, or update a dataslot value of a process instance in BP Server.
For more information on the Business Process Server related terms, see the Glossary of this guide. For the complete list of Business Process Server terminology, see the *OpenEdge Business Process Server: Terminology Guide*.

**Before you begin:**
Progress Software Corporation recommends that you use ABL Application Programming Interfaces (APIs) of Business Process Server to work with its functionalities because, OpenEdge release 11.3 onward, we do not support BP Server APIs. For more information on ABL APIs for Business Process Server, refer to the *OpenEdge Development: ABL Reference* guide.
This chapter lists the basic Business Process Server features.

For details, see the following topics:

- **BP Server features**
- **Duplicating a mapped name in an external workstep**
- **BP Server MDB control**
- **Archiver**

**BP Server features**

BP Server continues to support the following features:

- Managed adapters
  - Provides frequently used adapters as out-of-the-box managed adapters.
  - Provides the Adapter Configuration and Mapping Framework. For more information, refer to the Managed Adapters Guide.

- Process template versioning
  - Allows installation of a new version of an installed application.
  - Allows different versions of an application to be simultaneously installed, where at any time, a single version remains in the Installed state and all the others in the Deprecated state.
• Facilitates the dynamic creation of versions without stopping BP Server. For more information, refer to Process template versioning on page 65.

• Four-eye principle for performers
  • Allows different performers to be assigned to successive critical worksteps involving business decisions.
  • Facilitates excluding performers of one or more worksteps, for another human workstep. For more information, refer to Excluding performers for worksteps on page 73.

• Query service
  • Helps faster retrieval of the workflow information.
  • Reduces the network traffic and the load of the BP Server. Refer to Query service on page 79.

• Dynamic work item management using queues
  • Maintains the list of queues and its dynamic groups in a database. Provides for each queue to contain multiple dynamic groups and users.
  • Allows making work items available to other performers, if not completed by the specified due date. For more information, refer to Dynamic work item management using queues on page 135.

• Proxy user support
  • Allows delegation of all or some of the applications to one or more users or groups.
  • Provides preferred delegation setting that only needs to be applied as required.

• Java expressions in conditions and preconditions
  • Enhances the usability and power of the business processes.
  • Provides easier handling of BP Server dataslots (such as DECIMAL and DATETIMETZ) in decision conditions or preconditions.

• Time-bound activation
  • Allows a user to specify the time when a workstep can be activated, using the "Activate on" field in the workstep properties.
  • Provides the additional property "Activate after" to allow a user to specify the time after which the workstep can be activated.

• Callback adapter
  • Sends alerts when the Business Process Servers change state.
  • Facilitates quick action by the administrator for server problems. For more information, refer to the Application Developer’s Guide.

• Completed work item performer tracking
  • Records the actual performer of a completed work item in the database.
  • Provides the actual performer facilitating other dependant actions.
• Dynamic adapter loading
  • Facilitates modification of the Business Process Server adapter/external performer during runtime, so that these changes are reflected immediately without server restart.

• Extend due date function
  • Allows the user to specify a duration by which the due date of a workstep can be extended when it becomes overdue.

• Workstep process timer
  • Provides for the workstep properties to include timer information.
  • Executes the specified action after timeout. Refer to the section Setting Workstep Properties in the OpenEdge Getting Started: Developing BPM Applications with Developer Studio for more information.

• Global dataslots
  • Defined at the process template level and available for use with all the instances. They are shared and any update to the value is available to all the instances in real-time.
  • These dataslots are used as the workstep input and output dataslots for Activity, Adapters, and Subprocess worksteps.

• LIST dataslots
  • Allows you to define a list of choices for a dataslot in the Progress Developer Studio for OpenEdge. Multiple choices can be selected from the list.
  • The list contains a collection of non-unique restricted elements. For more information, refer to LIST dataslots on page 150.

• EJB method for atomic workstep
  • Performer decided at runtime.
  • Method plugged-in to obtain the performer(s). Refer to the EJB performer in atomic workstep on page 155 for more information.

• JMS based event publisher
  • BP Server events published to the external applications.
  • Default and user-defined Sender Implementation. Refer to the chapter JMS based event publisher on page 183 for more information.

• Check for BP Server
  • When BP Server is not running and the JMS message for the external workstep is received, the message is resent on the Queue three times, with a default waiting period of 15 seconds between each resend.
  • Immediately before and after the adapter execution, checks if BP Server is running. Adapter execution is halted if the server is not running.
  • Persistent JMS server ensures message delivery, hence restart external services is not called if either of external/Web services have persistent delivery mode.

• Read-only APIs
• Some APIs are read-only for performance improvements.

• Object-oriented API
  • Support for the object-oriented APIs without using stateful session beans. For more information, see Using BP Server APIs for process template management on page 96.
  • Use of the smart value objects to implement object-oriented APIs using stateless session beans. For more information, see Using BP Server APIs for process template management on page 96.
  • The smart value objects improve the performance and allow transparent communication with the BP Server components.
  • Changes to the data structure without affecting the client code, which is possible by using the generic attribute access pattern for the smart value objects and the API return data.
  • Support for collection operation on the ProcessTemplate, ProcessInstance, WorkstepInstance, WorkItems, andDataslots (Aggregate Value Object Pattern).
  • Support for sorting on the collection data.
  • Use of the session object to avoid duplicate authentication.

• Proxy workitem API
  • Gets the work item list for a performer, including the work items which are assigned to the performer as proxy.

• BP Server admin utility
  • Command-line utility.

### Duplicating a mapped name in an external workstep

BP Server provides a mechanism to define a mapped name for a dataslot in an external workstep. You can provide the mapping for both input as well as output dataslots. This facilitates the Adapter or external performer (EP) provider to have different method names from its input/output dataslot names. Though BP Server requires having unique names for each dataslot in a process, it does allow duplicate mapped names for multiple dataslots (two or more dataslots can have the same mapped name).

Note the following implications of duplicate mapped names:

• If an external workstep has a duplicate mapped name for either an input or output dataslot, then observe in the following method:

  ```java
  void setProcessContextData (Hashtable)
  ```

  If implemented by the Adapter / EP, then a few input/output dataslots which are mapped to the same mapped name are missing. This is because the mapped name is used as a key in the hashtable, which cannot be duplicated. Therefore, if ‘n’ dataslots have the same mapped name, then ‘n-1’ dataslots are missing.

• If an external workstep has a duplicate mapped name for input dataslots, then observe in the following method:

  ```java
  void setAllInputDataslots (Hashtable)
  ```
If implemented by the EP, then a few input dataslots which are mapped to the same mapped name are missing. This is because mapped names used as a key in the hashtable cannot be duplicated. Therefore, if ‘n’ input dataslots have the same mapped name, then ‘n-1’ dataslots are missing.

• If an external workstep contains a duplicate mapped name for output dataslots, then an EP must implement the following method:

    Hashtable getAllOutputDataslots ()

An EP cannot implement the individual getter method, because method overloading is not allowed based on different return types. Also, the hashtable returned by the above method is missing a few output dataslots, mapped to the same mapped name. This is because the mapped name is used as a key in the hashtable, which cannot be duplicated. Therefore, if ‘n’ output dataslots have the same mapped name, then ‘n-1’ dataslots are missing, which may result in unwanted dataslot updates.

### BP Server MDB control

The BP Server MDB control utility monitors the following services in the section below.

**Services monitored**

• BLRESUMESWS
• BLEMAIL
• BLBIZSTORE
• BLMCONSOLE
• BLSESSION
• CHECKDUE

If any of these services fail, then BP Server MDB Control sends a message to start another MDB to provide the same service.

**Setting the configuration parameters**

Set the following configuration parameters to start the BP Server MDB control utility:

• `bpserver.mdbcontrol.start` (true | false)
  
The parameter to start BP Server MDB control. The default value is “false”.

• `bpserver.mdbcontrol.schedule` (time in minutes)
  
The parameter that gives the schedule in the MDB control looks for any MDBs that have stopped. The default value is 5 minutes.
Archiver

Business Process Server provides Archiver, a sophisticated utility that schedules event archiving. With this utility, you can plan and schedule various archiving options and then Business Process Server automatically takes care of archiving them. By default, this utility does not start with the Business Process Servers and thus requires manual starting.

You can configure the Archiver from Business Process Portal Administration module. You can also edit the log level and the maximum number of backup archive files in the `eebpslog.conf` file. Refer to the Business Process Portal Administrator’s Guide for more information.
In Business Process Server, the due date for any task is calculated by adding the duration of the task specified in the process template, to the time the task is activated. Many applications need to take specific actions whenever a task is not completed by the due date. It means that the due date calculation must be accurate and realistic. The due date is calculated according to the working schedule of the performers of the task, and should take into account the business hours, business days, holidays, and weekends.

Business Process Server provides the Business Calendar feature where a number of business calendars can be defined enabling BP Server to compute the correct due date based on the system, group or user business calendars as applicable.

Business Calendar supports the following features:

- Business calendar definition using XML
- Multiple calendars across different time zones
- Country-specific format
- APIs for due date calculation
- Import and export utilities for the business calendar data
- J2EE compliance

For details, see the following topics:

- Business calendar tags
- Multiple calendars
- Calculating the due date
Business calendar tags

All the defined calendars are saved as XML files. This XML definition uses tags to identify the working and non-working time. Tags are defined for working time as well as for the non-working time.

Working time tags

The working time is defined in hours, using the 24-hour format.

Business hours

An example of the business hours definition is shown below:

```xml
<business-hours>
  <description>business hours</description>
  <business-hour start-time="0800" end-time="1700"/>
</business-hours>
```

The difference between the start and end time includes the lunch hour. Any task after the end time is assigned to the next business day. Similarly any task before the start time begins at the start time.

In some sectors, the working hours are different on different days. For example, in the banking sector in India the business hours are different on Saturdays compared to the other weekdays. Such non-standard business hours can be specified for a particular day in a week and/or for particular dates in a year as shown in the following example:

```xml
<!-for all Mondays -- >
  <business-hour start-time="1000" end-time="1400" dow="Mon"/>
<!-For 16th January, 2004 -- >
  <business-hour start-time="1000" end-time="1400" year="2004" month="Jan" day="16"/>
```

Non-working time tags

The non-working time can be lunch hour (Mid-day break), weekends, or holidays.

The lunch hour

An example of the lunch hour definition is shown below:

```xml
<lunch-hours start-time="1300" end-time="1400"/>
```

The lunch hour are excluded while calculating the duration / due date.

Half-day holidays

Sometimes only the first half or the second half of the business day is a holiday.
The start time/end time of a half day is decided by the lunch hour. The start time of the lunch hour marks the starting of a half day holiday for the second half. And the end time of the lunch hour marks the end of a half-day holiday for the first half. The lunch hour are excluded while calculating the duration and the due date.

An example of the half-day holiday is shown below:

```xml
<holiday name="HalfDay" first-half="true" month="Jan" day="10" />
```

This example specifies the first half of 10th January as a half-day holiday. The attribute second-half="true" can be used to specify that the second half of the day is a half-day holiday.

**Weekends**

The weekends may be different across countries and across business entities. For example, a typical American weekend is Saturday and Sunday, while the Arabian Gulf Countries have Friday and Saturday as weekends. In some business sectors, the weekly off days are only Sundays, while some offices may have second and fourth Saturdays as weekend days besides all Sundays.

The weekend definition therefore has two attributes - `dow` (day-of-the-week) and `occurrence`.

- **dow**: The attribute `dow` defines the day of the week with the possible values—Sun, Mon, Tue, Wed, Thu, Fri, Sat, as per [java.util.Calendar](https://docs.oracle.com/javase/7/docs/api/java/util/Calendar.html) definitions.
- **occurrence**: The attribute `occurrence` defines which occurrence of the day is a holiday. The valid values are—first, second, third, fourth, fifth, all. The default value is "all", indicating that all occurrences of the specified day are holidays. The value "last" is not valid for the weekends. You can compare it with the attribute `occurrence` in the case of [Fixed holidays](#) on page 32.

An example of the weekend definition is shown below:

```xml
<week-end>
  <holiday name="Sunday" dow="Sun" occurrence="first" />
</week-end>
```

The holiday name acts as a label, and can be "Sunday" or any other suitable name. Some more examples of the weekend format are shown below.

- **Only Sunday as the weekend day:**

```xml
<week-end>
  <holiday name="Sunday" dow="Sun" />
</week-end>
```

- **Both Sunday and Saturday as the weekend days:**

```xml
<week-end>
  <holiday name="Sunday" dow="Sun" />
  <holiday name="Saturday" dow="Sat" />
</week-end>
```

- **All Sundays and the first Saturday of every month as the weekend days:**

```xml
<week-end>
  <holiday name="Sunday" dow="Sun" />
  <holiday name="Saturday" dow="Sat" occurrence="first" />
</week-end>
```

- **All Sundays and the second and fourth Saturdays of every month as the weekend days:**

```xml
<week-end>
  <holiday name="Sunday" dow="Sun" />
  <holiday name="Saturday" dow="Sat" occurrence="second" />
</week-end>
```
Fixed holidays

A business calendar for any year always includes a number of fixed holidays. They could be date-specific holidays, such as the New Year Day on January 1, or the Independence Day on July 4. These holidays are specified with the attributes month and day.

Example

An example of the date-specific holiday is shown below:

```xml
<fixed-holidays>
  <holiday name="New Year Day" month="Jan" day="01"/>
  <holiday name="Independence Day" month="Jul" day="4"/>
  <holiday name="Christmas Eve" month="Dec" day="24"/>
  <holiday name="Christmas" month="Dec" day="25"/>
</fixed-holidays>
```

Some fixed holidays are day-of-the-week-specific, such as the Thanksgiving Day falling on the fourth Thursday of November. The date varies depending on the year, and hence these holidays are specified with the attributes month, dow and occurrence.

An example of the day-of-the-week-specific holiday is shown below:

```xml
<fixed-holidays>
  <holiday name="Thanksgiving Day" month="Nov" dow="Thu" occurrence="fourth"/>
</fixed-holidays>
```

Note that in this case, the attribute occurrence has valid values - first, second, third, fourth, fifth, last. The value "fifth" refers to the fifth occurrence in the month, if any. If you select "last", then the last occurrence in the month is considered — it could be the fourth occurrence or the fifth. The value "all" is not valid for the fixed holidays. You can compare it with the attribute occurrence in the case of Weekends on page 31.

Year-specific holidays

Year-specific holidays are always qualified for the specified year.

Example

An example of the year-specific holiday is shown below:

```xml
<holidays year="2011">
  <holiday name="Company Annual Event" month="Nov" day="18"/>
</holidays>
```

The above example shows that the Company Annual Event takes place on a different date each year, and in the year 2011, it falls on November 18. The year-specific holidays can also be day-specific, as explained in the case of Fixed holidays on page 32.
Note: Some year-specific holidays can be half-day holidays. Refer to Half-day holidays on page 30.

Even when a particular year has no year-specific holidays, you must define the empty tags as shown below:

```xml
<holidays year="2011"/>
```

This is the only way to indicate that calendar entries are to be generated and persisted for the year 2011. You can specify working and non-working time for a range of years also. You need to add a separate year tag for each of the year of that year range. An example of year range tags for the year range from 2011 to 2014 is shown below.

```xml
<holidays override="true" year="2011"/>
<holidays override="true" year="2012"/>
<holidays override="true" year="2013"/>
<holidays override="true" year="2014"/>
```

### Time zone and locale tags

The Business Calendar feature in Business Process Server supports multiple calendars in different time zones with different locales. The tag `<locale>` has attributes country and `language`. For the tag `<timezone>` you need to define the attribute `zoneid`.

#### Example

An example of the `<locale>` and `<timezone>` tags are shown below:

```xml
<calendar name="sbmcal">
  <locale country= language=/>  //language is optional
  <timezone zoneid=/>
</calendar>
```

Both the tags `<locale>` and `<timezone>` are optional. When not specified, the default values are used.

When a specific locale and time zone is specified, the Business Calendar invokes the `java.util.Calendar` constructor that takes both `locale` and `timezone` objects.

Based on the locale or time zone specified, the Calendar object outputs different values. For example, the method `calobj.getFirstDayOfWeek()` returns the first day of the week specific to a country, which is Sunday in USA, but Monday in France.

When you specify a time zone, the Calendar object takes care of the time offset automatically.

The following example shows how the various attributes are specified.

```xml
<locale country="US"/>
<locale country="US" language="en"/>
<timezone zoneid= "America/Los_Angeles"/>
```

### Multiple calendars

Business Process Server supports multiple calendars for individual users and groups, in addition to the default system calendar.
**System calendar**

When the Business Process Server administrator creates a particular calendar, the calendar can be set as the default system calendar. The SBM_CALENDAR table records whether a particular calendar is a "system" calendar or not. The different calendars associated with the users and groups use separate relationship tables.

When a new user or group is created, the default calendar for the newly created user or group is the system calendar. The Business Process Server administrator sets the calendar for the users and groups. Some users may or may not have their own calendars.

**Note:** If a user does not want to use the calendar function, then in the configuration file `bpserver.conf`, the parameter `bpserver.calendar` can be set to "false".

---

**Precedence for calendars**

A user calendar always has the highest precedence. When a user calendar is available, any due date calculation for that user always considers the user calendar.

When an activity workstep has a single performer, then the due date for the workstep and workitem is calculated using that individual user’s calendar, if available; otherwise, it is calculated using the system calendar.

When an activity workstep has multiple performers (a group or a queue), and the workstep is marked as 'perform by any member', then both, the workstep instance’s and work item’s, due dates are calculated using the system calendar.

When an activity workstep has multiple performers (a group or a queue) and the workstep is marked as 'perform by all members', then the due date for individual work items is calculated using individual user’s calendar, if available; otherwise, it is calculated using the system calendar.

When there is no system calendar, the BP Server calculates the due date by adding the duration to the workstep instance activation time.

---

**Calculating the due date**

A business calendar in XML format is parsed and stored in the database. In the SBM_CALENDAR table, the calendar name and the XML are also persisted.

The Calendar parser utility parses the XML and persists all holidays and weekends in the SBM_HOLIDAY table.

The database properties can be specified in the business calendar XML, or using APIs. Using these properties, the Business Calendar creates the database connection and persist all the required calendar information.

When the database properties are not specified either through an API or an XML, then the calendar utility tries to read the database properties from `oebpsdb.properties`. The Business Calendar reads `oebpsdb.properties` only using classpath.

**Note:** For calendar runtime initialization SBMCalendar.init(), if the specified properties are invalid, then an exception is raised.
For every due date calculation, a BizCalendar object is created and passed on to the getDueDate(BizCalendar) of the SBMCalendar object which returns the due date as a Calendar object. The due date is returned as a java.util.Calendar object so that information about locale, timezone, and other helper methods are available to the user.

```java
long startDate = (new java.util.Date()).getTime();
long duration = 5*24*60*60*1000; //5 days in milliseconds
SBMCalendar sbmcal = new SBMCalendar("savvion-usa-calendar.xml");
BizCalendar bcal = new BizCalendar(startDate, duration);
Calendar duedate = sbmcal.getDueDate(bcal);
System.out.println("DUEDATE: " + duedate.getTime().getTime());
```

Note: Cached calendars are updated during BP Server restart.

---

**Workstep instance due date**

The workstep instance due date is always based on the group calendar or the system calendar and the estimated duration defined for the workstep.

- If the workstep has a single performer as group:
  - If the group calendar is defined, then it is used.
  - If the group calendar is not defined, then the system calendar is used.

- If the workstep has more than one performer, then the system calendar is used.

The calculation of the due date of the workstep instance never uses the user calendars, as individual calendars may vary in vacation days and working hours. Note that at the design stage, the application design considers only standard time and duration and not any individual variation.

**Pre-script in workstep due date**

The workstep instance due date available in the pre-script is the due date based on the system calendar. Since the pre script can modify the datlaslot value used as the performer, the due date cannot be based on the performer before pre-script execution.

Once the pre-script is executed, BP Server updates the workstep instance due date appropriately based on the calendar of the performer if it happens to be a single Group.

**Updating due date in pre-script**

When the due date of the workstep instance is updated in pre-script, BP Server does not recalculate the due date based on the Group calendar, even if there is a single Group performer. Hence the due date specified by the user remains the same.

**Retrieving due date in pre-script**

Within the pre-script, the following APIs can be used to access the due date:

- `var dueDate = jst.getDueDate();`

  This API gives the present due date of the workstep instance that has been calculated using the system calendar.
var dueDate = jst.calculateActivityDueDate();

This API calculates and provides the due date based on the group calendar using the current performer information.

**Work item due date**

BP Server uses the business calendar for due date calculation only at the time of work item creation for the work step type Group Any and Group All.

Work item due date is not recalculated when

- Work item state changes from available to assigned.
- Work item is reassigned.
- Tasks are delegated.

When a work item is created for a user, the highest precedence is given to the calendar assigned to the user. If the user has no calendar, then the workstep instance due date is assigned to the work item. As described before, the workstep instance due date may be based on either system or group calendar.

In a Group All workstep, each work item may have a different due date based on the individual user calendars. However, the workstep instance due date is not adjusted to any of the work item due dates. That is because the workstep instance due date always considers the estimated duration and not the individual calendars.

The difference between work item due date and workstep instance due date can be used very effectively to analyze and improve the estimated duration as well to identify the reasons for the performers who take longer duration compared to the estimated duration.

Note that Timer Actions could be used to automate reassigning all work items that take longer duration to achieve effective time management.

**Calculating the duration**

The calculation of due date requires the start time and the duration. Whether the duration specified by the user is in seconds, minutes, hours, days, months or year, BP Server converts it into milliseconds. This duration in milliseconds is used to calculate the due date using the specific calendar.

BP Server supports the duration calculation using the Duration object. In this implementation for converting the days into hours, a factor of 24 is used. Note that this factor does not consider working hours as 8.

For converting months into days, a constant factor of 30 is used irrespective of the month. For converting years into days, a constant factor of 365 is used even for a leap year. Thus, the "month" and "year" conversions are not exact and might lead to slight digression from the due date expected by the user. However, users can use other duration units, such as "days", "hours", "minutes", and "seconds", to accurately specify the time.

For example,

```java
Duration duration = new Duration("6d");
System.out.println("time:" + duration.getTime());
```

The `getTime()` method returns the duration in milliseconds.

If the duration specified by the user is 6 days, then it does not correspond to 6 business days. The duration is converted to hours, using 6 times 24 hours giving a value of 144 hours. If one business day equals 8 hours, then this duration amounts to 18 business days.
Working hours for regular business hours

When a calendar with regular business hours is used for calculating the due date then, the working hours are calculated as:

\[(\text{Regular End Time} - \text{Regular Start Time}) \text{ minus } (\text{Regular Mid-day Break End Time} - \text{Regular Mid-day Break Start Time})\]

For example, consider a calendar with

- Regular Start Time: 09:00
- Regular End Time: 18:00
- Mid-day Break Start Time: 13:00
- Mid-day Break End Time: 14:00

The working hours are calculated as \((18:00 - 09:00) - (14:00 - 13:00)\), equalling 8 hours.

Working hours for special business hours

For special business hours, the working hours are calculated considering the regular Mid-day break Start and End time for that particular calendar.

Typically, it is calculated as:

\[[\text{Special End Time} - \text{Special Start Time}]\]

minus

\[\text{earlier of (Regular Mid-day Break End Time) or (Special End Time)}\] minus \[\text{later of (Regular Mid-day Break Start Time) or (Special Start Time)}\]

Consider a calendar with:

- Regular Start Time: 09:00
- Regular End Time: 18:00
- Mid-day Break Start Time: 13:00
- Mid-day Break End Time: 14:00

The Table 2: Special business hours examples on page 37 illustrates the use of Mid-day Break hours.

Table 2: Special business hours examples

<table>
<thead>
<tr>
<th>Special business hours</th>
<th>Duration</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start Time</td>
<td>End Time</td>
<td></td>
</tr>
<tr>
<td>07:00</td>
<td>13:30</td>
<td>((13:30 - 07:00) - (13:30 - 13:00)) 6 Hours</td>
</tr>
<tr>
<td>07:00</td>
<td>13:00</td>
<td>((13:00 - 07:00)) 6 Hours</td>
</tr>
</tbody>
</table>
## Working hours for half-day holiday

For half-day holiday, the business hours are of two types:

- First-half holiday: Mid-day Break Start Time to Regular End Time.
- Second-half holiday: Regular Start Time to Mid-day Break End Time.

However, Mid-day Break hours are excluded while calculating the actual working hours.

For the first-half holiday, working hours are calculated as:

\[(\text{Regular End Time} - \text{Half-day Start Time}) - (\text{Mid-day Break End Time} - \text{Mid-day Break Start Time})\]

which works out to be \((\text{Regular End Time} - \text{Mid-day Break End Time})\)

For the second-half holiday, working hours are calculated as:

\[(\text{Half-day End Time} - \text{Regular Start Time}) - (\text{Mid-day Break End Time} - \text{Mid-day Break Start Time})\]

which works out to be \((\text{Mid-day Break Start Time} - \text{Regular Start Time})\)

Consider a calendar with:

- Regular Start Time: 09:00
- Regular End Time: 18:00
- Mid-day Break Start Time: 13:00
- Mid-day Break End Time: 14:00

The Table 3: Half-day working hours examples on page 39 illustrates the use of Mid-day Break hours.
Table 3: Half-day working hours examples

<table>
<thead>
<tr>
<th>Start Time</th>
<th>End Time</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>09:00</td>
<td>14:00</td>
<td>(14:00 - 09:00) - (14:00 - 13:00)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4 Hours</td>
</tr>
<tr>
<td>13:00</td>
<td>18:00</td>
<td>(18:00 - 13:00) - (14:00 - 13:00)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4 Hours</td>
</tr>
</tbody>
</table>

**BP Server API for due date calculation**

Sometimes the due date of a workstep instance or work item needs to be changed.

**APIs provided**

BP Server already provides the following APIs to change the due date value.

- `WorkStepInstance.setDueDate(DateTime newDueDate)`
- `WorkItem.setDueDate(long newDueDate)`

With the Calendar support, BP Server now provides the following additional APIs to calculate the due date using the specified business calendar or using the system calendar.

- `Calendar WorkStepInstance.getBizDate(String calName, long startTime, long duration)`
- `Calendar WorkStepInstance.getBizDate(long startTime, long duration)`

If no calendar name is provided, then the API uses the system calendar. If there is no system calendar found, then the result of addition of startTime and duration is returned.

For WorkItem Smart Value Object (SVO), similar APIs are provided.

**BizCalendar SVO**

For every due date calculation, an object of the class BizCalendar is created that accepts the start-time and duration. The BizCalendar object is then passed to the `SBMCalendar.getDueDate(BizCalendar)`, for the due date calculation. This SVO is used to pass on the information to the `SBMCalendar.getDueDate()` method and is also used as a placeholder during the calculation.

BizCalendar also handles duration in string format such as "5d30m30s" using the specified Duration object.

```java
Duration durObj = new Duration("5d30m30s");
long startDate = (new java.util.Date()).getTime();
BizCalendar bcal = new BizCalendar(startDate, durObj);
Calendar duedate = sbmcal.getDueDate(bcal);
```

BizCalendar supports duration with the formats YEAR(y), MONTH(m), DAY(d), HOUR(h), MINUTE(m), SECONDS(s). The Table 4: Duration format usage on page 40 shows the use of these formats for Duration.
Table 4: Duration format usage

<table>
<thead>
<tr>
<th>Duration format</th>
<th>Actual duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>1y10M25d5h30m3s</td>
<td>1 year, 10 months, 25 days, 5 hours, 30 minutes, 3 seconds</td>
</tr>
<tr>
<td>25d5h30m</td>
<td>25 days, 5 hours, 30 minutes</td>
</tr>
<tr>
<td>5h30m</td>
<td>5 hours, 30 minutes</td>
</tr>
<tr>
<td>5h</td>
<td>5 hours</td>
</tr>
<tr>
<td>30m</td>
<td>30 minutes</td>
</tr>
</tbody>
</table>

```
Duration dur = new Duration("1y10M25d5h30m3s");
long durval = dur.getTime();
```

This example returns 59635803000 milliseconds.
Process refresh

Quite often, within an organization, some internal or external changes prompt for changes in a process. For example, if a corporate office is moved, then the return address for all outgoing mails changes. In such a situation, it becomes necessary to replace the existing process by a modified process. Typically, this calls for versioning, as explained in Process template versioning on page 65. However, when the older versions need not be maintained, the installed process can be directly replaced, with the running process instances adapting seamlessly to the new process.

**Note:** During process refresh, do not to carry out any activity until the refresh is completed.

For details, see the following topics:

- Changes during refresh
- API from ProcessTemplate SVO
- BP Server admin commands for Process Refresh feature

Changes during refresh

Since it is necessary to maintain the integrity of the processes and the system, only the following changes are allowed during refresh:

- Add worksteps.
- Modify selected workstep properties.
Chapter 4: Process refresh

- Add instance level and global dataslots.
- Modify selected dataslot properties.
- Modify selected link properties.

**Note:** If a process requires changes other than those allowed for process refresh, then it is recommended to version the process template as described in Process template versioning on page 65.

After Process Refresh, changes in the priority, performer and instructions workstep properties are not effective for the process instances which are already activated, if the said workstep was already activated once or more before the process refresh operation. For the process instances created after process refresh operation, or if the said workstep is activated for the first time only after process refresh operation, then the changes made in the above mentioned properties are effective immediately.

**Adding worksteps**

You can add worksteps subject to the following conditions:

- If a new workstep is added, then the target cannot be an existing And-join type workstep.
- For all existing And-join worksteps, the source worksteps should not be modified.

By default, BP Server does not allow addition of new And-join workstep for the Process Refresh feature. In specific situations, you can allow the addition by setting the following property in the file `bpserver.conf`:

```
bpserver.ptreplace.safemode=false
```

When this property is set to "false", BP Server allows the addition of new And-join worksteps. Note that this property is not present in the `bpserver.conf` by default. If you need to set it, then you have to enter it in the conf file and then set it as desired. However, this change in property is very critical and we recommend it to be done with caution.

Consider a process in which the workstep `Activity3` is activated. With Process Refresh, you add a new workstep `Activity4` and an And-join such that `Activity3` and `Activity4` terminate in the And-join. In this scenario, instances from the source process template never has the `Activity4` workstep activated and hence the flow does not proceed.

As a best practice, it is recommended not to add any new And-join workstep.

**Note:** Deleting worksteps is not allowed.

**Workstep properties**

The following sections describe properties you can modify in BP Server.

**General properties**

- Label
- Performer — change effective only at the next activation
Note: In the case of Group Performers, change from ANY in the Group to ALL in the Group and vice versa, is not allowed.

- Description
- Duration — change effective only at the next activation
- Priority — change effective only at the next activation

**Fields properties**

- Instructions — change effective only at the next activation
- Link instructions to dataslots — change effective only at the next activation
- Options — Show instructions, Show priority, Show duedate, Show start date
- Dataslots — add or delete for input and output dataslots

Note: For the subprocess and adapters worksteps, if the process instance is activated in the above worksteps, then the output dataslots continue with the old process. When the workstep is activated the next time, it is from the new process.

- Move up or Move down for input and output dataslots
- Edit dataslot label, edit options — Editable and Required
- Enable reassign option

**Advanced properties**

- Activate On — change effective only at the next activation
- Activate After — change effective only at the next activation
- Pre Javascript — change effective only at the next activation
- Wait for condition — change effective only at the next activation

Note: If the precondition is removed, then the workstep is suspended, and can only be resumed from BP Server Admin.

- Use custom form for Activity workstep
- Exclude performer — add, delete, and edit: change effective only at the next activation
- Activity workstep e-mail support — change effective only at the next activation
- Post Javascript
- Overdue/timer actions — add, delete, edit, move up and move down: change effective only at the next activation
- On last overdue — change performer and complete workstep
Note: If you set the change performer action on the last overdue workstep, then ensure that an individual user is assigned to such a workstep. If a CHARACTER dataslot is assigned to such a workstep, then it must contain value of the user name. If a human activity task is set as 'perform by all members' of a group, then all those workitems are reassigned to this user.

Rollback properties

For a process with rollback enabled:

- The flow cannot be changed.
- New worksteps cannot be added.
- The rollback point cannot be changed.
- The workstep with "activate rollback point action" cannot be changed.

For any existing workstep:

- The rollback point cannot be added.
- Activate rollback point action cannot be added.

Adding dataslots

You can add instance level and global dataslots, and define their default values. In the case of LIST and Object type of dataslots, the default value is taken as null.

Note: If you add any new dataslots during process refresh, then you must restart the ejbserver.

Deleting dataslots is not allowed.

Modifying dataslot properties

You can change the following properties of dataslots:

- Default format type
- Default values for instance level dataslots: change effective only at the next activation

Note: If you change the default value of a global dataslot, then a warning is logged in the bpserver.log file, and the existing value is used.

- Label
- Choices for CHARACTER dataslots: can be added but not edited or deleted
- Options: Editable and Required
- Management Access
- Tasks column

The dataslot's initial value is reflected only on the first or next activation.
Modifying link properties

You can change all the properties of links.

API from ProcessTemplate SVO

The following APIs are provided from the ProcessTemplate SVO.

APIs provided by ProcessTemplate SVO

• public HashMap validateForUpdate(String xmlfile)
  wherein xmlfile can be the absolute path to the new XML file, or the entire XML file as String.
  HashMap contain the following keys:
  • RESULT — the value is "true" or "false".
   If the value of RESULT is "false", then the additional keys are traversed to get the information why the validation failed.
• public void replace(String xmlfile)
  wherein xmlfile can be the absolute path to the new XML file, or the entire XML file as String. If the xmlfile is null, then the process XML is obtained from the
  OEBPS_HOME\ebmsapps\<processname>\processtemplates\processname.xml, where <processname> is the name of the process.

BP Server admin commands for Process Refresh feature

Typically, the process refresh is carried out from Progress Developer Studio for OpenEdge using the Refresh process template option during the process publishing. You can also validate and refresh using the BP Server Admin utility.

BP Server Admin provides the following commands for the Process Refresh feature:

• validateProcessForUpdate ptname
  This command is used for validating the process to be refreshed.
  validateProcessForUpdate takes the process name which is refreshed. The new process XML can be specified as the absolute location for the XML file or the entire XML file as String. If it is not specified, then the API reads from the
  OEBPS_HOME\ebmsapps\<processname>\processtemplates\processname.xml, where <processname> is the name of the process.
  Examples:
  validateProcessForUpdate ptname c:/temp/refreshpt.xml
  validateProcessForUpdate ptname
  validateProcessForUpdate ptname refreshPtXmlDefinitionAsString
If the command `validateProcessForUpdate` does not show any errors, then the process can be refreshed.

- **replaceProcess ptname**

  This command is used for refreshing the process without versioning.

  replaceProcess takes the process name which is refreshed. You must copy the modified file to `OEBPS_HOME\ebmsapps\<ptName>\processstemplats` folder.

  While refreshing, the `ptName` is used and the corresponding modified file is taken from `OEBPS_HOME\ebmsapps\<ptName>\processstemplats` folder.

**Note:** When the process is refreshed successfully, any changed adapters and the interfaces changes for the Activity worksteps and all the process dependent changes must be copied to the `OEBPS_HOME\ebmsapps` folder and also to the portal server `ebmsapps` folder.
Process instance migration

In a dynamic organization, processes continuously keep changing. In many cases, the cost of ownership of processes can be significantly controlled if changes can be applied readily to the running instances of processes. Business Process Server provides the feature of Process Instance Migration, allowing for bulk migration of instances of processes from one version to another.

A business process instance can be migrated to another version only when all its active workstep instances have human performers. If a business process instance has any active workstep of type Adapter or WebService or Subprocess, then the instance cannot be migrated to the new version.

Note: Once the instances are migrated to the target, the instances in the source template still exist. However, they are suspended. You can undo the process instance migration if you know the ID of the migration. Refer to the Business Process Portal Administrator’s Guide for more information.

For details, see the following topics:

- Migration features
- Migration architecture
- Migration mapping
- Filters for migration
- Migration run
Migration features

Instance migration is carried out by mapping worksteps and dataslots between source (existing) and target (new) processes. The main features are described in the following sections.

Process instances

• The list of instances to be migrated can be specified explicitly with the list of instance IDs. It is also possible to get the list of qualified instances by using the default filter conditions and also with the user-defined filter conditions.

• The migrated process instance name contains the original process instance name suffixed with the constant "#IM#" and followed by the new process instance ID.

• Dynamically modified process instances cannot be migrated. For more information about dynamic process instances, refer to Working with dynamic process instances on page 108.

• Process instances that are subprocess of other process instances are not migrated.

• Migration of process instances with rules is not supported.

• When you migrate a process instance from a source process template to a target process template, the events of the source process instance are not migrated. As a result, the target process instance’s audit history cannot show the worksteps that were completed in the source process instance before migration. Also, no audit history is displayed for the target process instance immediately after migration, because there are no events present in BPM Events table for the target process instance. The audit history for the target process instance can be seen only after some BP Server operations (like completing a workstep) are performed on it.

Worksteps of process instances

• The process instances having all the active worksteps with human performers can be migrated. Process instances with active worksteps with performers of type other than human performer, cannot be migrated.

• Process instances with one or more active workstep instances of type ANY cannot be migrated.

• One workstep of the source process can be mapped to either one workstep of the target process or none. One source workstep cannot be mapped to multiple destination worksteps.

• Only the worksteps of the same type can be mapped. For example, an adapter workstep can be mapped only to another adapter workstep.

Work items

• All incomplete work items of a process instance are not migrated.

• When a business process is migrated to a newer version, the worksteps assigned to human performers are reactivated and work items are created again based on the performer of the new process template.
For example, consider a process instance with an active human workstep of type ALL with 10 work items created and seven work items already completed before the migration. When this instance is migrated to a newer version, the migration tool ignores all the previously created and completed work items and creates 10 work items again, assuming that the workstep performer remains the same.

**Dataslots**

- All new dataslots for the migrated instances are updated with the default value specified in the template.
- Dataslot size changes are supported only for CHARACTER. The migration tool allows size changes only when the target dataslot size is larger than the source dataslot size.
- One source dataslot can be mapped to multiple target dataslots, or not mapped at all.
- Only the dataslots of the same type can be mapped. For example, a CHARACTER dataslot can be mapped only to another CHARACTER dataslot.
- In the case of dataslot type INT64, the following additional mapping is supported:
  - INT64 to CHARACTER
  - INT64 to INTEGER
  - INTEGER to DATETIME-TZ
Migration architecture

The Figure 1: Migration architecture on page 50 describes the architecture and the process of migration.

Figure 1: Migration architecture

Migration mapping

While migrating, you must map workstep as well as dataslots between the source template and the target template. For each map configuration, there is one XML that includes both workstep-mappings and dataslot-mappings of the source template and the target template.
Workstep mapping

A migration configuration includes workstep mappings and dataslot mappings. Workstep mapping from the source template to the target template are described below. It is also possible to define only the changes. For example, as shown in Figure 2: Workstep mapping on page 51, you need to provide the workstep mapping only for worksteps that are removed, newly added, or where the workstep name is changed.

Figure 2: Workstep mapping

Business Process Portal captures the workstep mapping information and generates the corresponding XML markup as follows:

```xml
<WorkstepMap>
  <Workstep source="Workstep2"/>
  <Workstep source="Workstep3" target="NewNameWorkstep3"/>
  <Workstep source="Workstep4"/>
</WorkstepMap>
```

The XML markup captures only the changes for worksteps, and the newly added worksteps in the target process template are not included in it.

Dataslot mapping

In dataslot mapping, the changes include dataslot name changes, type and size changes, removed dataslots in the source, and also the newly added dataslots in the target process template. The newly added dataslots in the target template are required for updating the newly added dataslot values in the database table with the default value specified.
The Figure 3: Dataslot mapping on page 52 shows a sample dataslot mapping.

**Figure 3: Dataslot mapping**

![Dataslot mapping diagram](image)

Business Process Portal and other applications capture this information and generate XML markup as follows:

```xml
<DataSlotMap>
  <DataSlot source="Dataslot1" target="Dataslot3"/>
  <DataSlot source="Dataslot2"/>
  <DataSlot source="Dataslot3" target="Dataslot7"/>
  <DataSlot source="Dataslot4"/>
  <DataSlot target="Dataslot9"/>
</DataSlotMap>
```

If the target process template includes a dataslot, process instances of the target process template will have the default value.

**Filters for migration**

While selecting instances for migration, you can define your own filters to ensure the validity of the migrated instance. For example, if an active workstep in the source process instance joins to a newly added AND workstep in the target process instance, and if one of the input to an AND workstep is already completed before or it is a newly added workstep.

BP Server adds a default filter condition that the source instance can be migrated only if all the active workstep instances have human performers. You can also add additional filter conditions on workstep instance attributes and dataslot values in Business Process Portal. The Filter XML file is generated internally and is stored in the database when a filter is created in Business Process Portal. An example of the Filter XML markup is shown below:

```xml
<InstanceFilter name="PartPurchase_v1_filter1">
  <Application source="PartPurchase_v1">
    <Condition>
      BLWSI.workstep_name in ('ws1','ws3') AND BLWSI.state != 15
    </Condition>
  </Application>
</InstanceFilter>
```

Note that the filter condition should be SQL92 compliant. A filter condition may include conditions on workstep instance attributes, instance dataslot values, and global dataslot values. The Table 5: Aliases for BP Server tables on page 53 describes the aliases for BP Server tables that can be used in this SQL92 complaint query. Note that these prefixes are the same as those defined in QueryService.
Table 5: Aliases for BP Server tables

<table>
<thead>
<tr>
<th>Table name</th>
<th>Prefix</th>
</tr>
</thead>
<tbody>
<tr>
<td>BP Server_WorkStepInstance</td>
<td>BLWSI</td>
</tr>
<tr>
<td>Instance Dataslot table</td>
<td>BLIDS</td>
</tr>
<tr>
<td>Global Dataslot table</td>
<td>BLGDS</td>
</tr>
</tbody>
</table>

When the specified conditions do not involve any instance dataslots or global dataslots, BP Server constructs the query without joining to the instance and global dataslot table for performance reasons.

When a filter is attached to a migration run, the SQL92 compliant condition specified in the filter object is appended to the above query. The default condition of BP Server and the user-specified conditions through the filter object together select the qualified instances for migration.

Migration run

For a migration run, all instances migrated in a batch should be identified. This is required for further events to undo the entire migration, or to display the list of instances migrated at different times. The migration tool generates a unique migration ID for each run. This migration ID is used to identify all instances migrated in a particular run.

Figure 4: Migration details on page 53 shows the migration details recorded for each migration run.

Migration commands

You can use the following commands to perform migration and undomigration operations. The specified user name and password should be of a valid BP Server account.

- `instancemigration.cmd -u <user_name> -p <password> -mapid <id> -filterid <id>`
- `instancemigration.cmd -u <user_name> -p <password> -mapid <id>`
• undomigration.cmd -u <user_name> -p <password> -migid <migration_id>

• undomigration.cmd -u <user_name> -p <password> -migid <migration_id> [-pid <source_process_instance_id>]

Note that whenever a set of process instances are migrated to another process template, a new set of process instances are created based on the original process instance data and migration map information. As a part of migration, the original set of process instances are suspended.

When you perform undomigration, the original process instance, which was suspended as a part of migration, is resumed and the newly created process instance is deleted. Therefore, for ‘undomigration,’ it is important that original process instances in the suspended state and the Migration Map used in the migration activity exist in the system. If any one of these two objects is deleted or removed, then ‘undomigration’ fails.

**Restoring to old process instance**

Though each instance migration is done in one logical transaction, the migration tool can undo the entire migration any time. A specific migration run is identified by a migration ID. The migration tool uses the PI_MIGRATION_INFO and PI_MIGRATION_DETAIL tables to undo all the migrated instances of a particular run.

It is also possible to undo the migration of a specific process instance. Whenever a migrated instance is restored to its original state using the "undo" operation, the status of the migrated instance is marked as "undo" so that it cannot be restored again. However, an instance which is already restored can be migrated again using a separate run.
Fetching the next available work item

This chapter explains how the fetch next available work item feature works.

For details, see the following topics:

- Fetch next available work item methodology

Fetch next available work item methodology

In BP Server, an activity workstep can have a user group as a performer. If "Any" is specified for the group performer, then the resultant work item is in an "Available" state. If any member of this group wants to work on this work item, then it is "Assigned" to this member and becomes not available to the other members of the group.

In a typical business scenario like a Call Center, there are hundreds of work items available to a group of operators. Business Process Portal displays all the available work items when a group member logs in. This makes the user experience very slow if large number of work items exist. Moreover, all the group members see all the work items and there are many conflicts when they pick up the same work item.

To avoid possible performance issues, BP Server provides the feature of fetching the next available work item from the pool of available work items.
**The steps are as follows:**

1. Multiple users call the session bean method.
2. Begin transaction and one thread (user call) obtains the lock.
3. Get the work items as per filter criteria from database. By default, only one work item is fetched, but you can increase it up to five work items.
4. Assign the work items to this user.
5. Return the work items to the user and end the transaction, releasing the lock.

   This scenario is also valid when the performer is a queue.
File class loader

BP Server supports dynamic loading of class files on demand to configure features as required.

The file class loader utility is capable of loading the class files from absolute locations as well as JAR files on demand. It can also load all the classes for specific applications from specified locations and from common locations as well. The class loader is flexible and can be configured to enable or disable features as required.

The `FileClassLoader` offers the following features:

- Application class loader — loads all the classes associated with an application using the same instance of the `FileClassLoader`.
- Loading class files from common resources — All instances of the `FileClassLoader` are capable of loading classes from common resources, other than the resources specific to them.
- Loading classes from JAR files — The `FileClassLoader` is capable of loading the classes from JAR files.
- Dynamic class loading — If a class is modified, then the `FileClassLoader` can load the new class definition.

For details, see the following topics:

- Class loading
- Resource loading
- Logging and debugging

Class loading

In general, the loading of a class by any class loader comprises the following steps:
1. Read the bytes of the class.
2. Define the class by interpreting the bytes of the class as per the rules of the JVM. This step identifies whether the class definition is correct or not.
3. Resolve the class - link the specified class.

Any custom class loader should extend the `java.lang.ClassLoader`. The `FileClassLoader` loads the class as follows:

1. If the class is a standard Java class, then let the parent class loader load it.
2. Otherwise, get the class from cache, if it exists and return.
3. If the class is not in the cache, then locate it physically and get its bytes.
4. If the class is not found, then let the parent class loader attempt to find it.
5. If the class is found, then define the class.
6. Cache the loaded `Class` object and return the same.

**Note:** According to the rules of the JVM, a class cannot be loaded by the same instance of the `ClassLoader` more than once. Therefore, if a class is modified and needs to be loaded again by the `FileClassLoader`, then a new instance of `FileClassLoader` needs to be created. The library supports dynamic class loading. If the same instance of `FileClassLoader` is used to load a class (which is modified) again, then a message is logged and the old class definition is returned.

### Resource loading

Business Process Server custom class loader also implements the following methods of the `java.lang.ClassLoader` class:

- `getResource(String name)`
- `getResourceAsStream(String name)`

The rules for loading the resources are the same as those for loading the classes.

Typically, when BP Server loads an adapter class from a JAR file in the `application/lib` folder, the resources contained in the JAR file also need to be loaded for the adapter. If Business Process Server custom class loader is used to load the adapter class, then the `getResource` implementation of the custom class loader is also able to load the Resources.

### Logging and debugging

Class loading is a complex process, and hence logging is required at all the important stages to debug any issues in the development or production environment. The logger used with this library is the `SBMLogger` from `com.savvion.sbm.util`. The logger is a configurable component, and the caller can set its own logger in the library. Therefore, whether it is a BP Server user or a BPM Events user, they can provide their own loggers to get the logging in their respective log files. Even if the logger is null, standard output is used for logging.

Similar to the logger, the debug flag is also configurable and can be provided by the caller along with the logger.
**Note:** The *FileClassLoader* library has been developed to work independently, without using Business Process Servers. The only dependency is on SBMLogger for logging purposes.
Dynamic class loader for adapters

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. BP Server provides dynamic class loading for adapters with the following features:

- Supports Application Class Loader and Adapter Class Loader.
- Class files can be loaded from multiple locations.
- Class files can also be loaded from JAR files.
- By default, application class loader is supported. In application class loader, all the adapters of an application are loaded by the same class loader.

The check "if class is modified" is based on the last modified time of the class file. If the class is in a JAR file, then the check is based on the last modified time of the JAR file as well as the last modified time of the class file.

BP Server allows loading class files from specific locations as well as from JAR files.

- Locations common to all applications
  - OEBPS_HOME\ebmsapps
  - OEBPS_HOME\ebmsapps\common\classes
  - OEBPS_HOME\ebmsapps\common\lib

- Locations specific to an application
  - OEBPS_HOME\ebmsapps\<application name>\lib

In general, BPM Events can load the class files placed directly in the following locations:
The classes from JAR files are loaded from the following locations:

- OEBPS_HOME\ebmsapps
- OEBPS_HOME\ebmsapps\common\classes

The classes from JAR files are loaded from the following locations:

- OEBPS_HOME\ebmsapps\<application name>\lib
- OEBPS_HOME\ebmsapps\common\lib

The ebmsapps\common\lib location is used to search for JAR files as well as class files, but the location ebmsapps\<application name>\lib is used to search only for the JAR files.

For details, see the following topics:

- Searching class files
- Controlling dynamic class loader
- Modifying the JAR files
- Loading adapter classes
- Dynamic class loader APIs

**Searching class files**

The class files are searched in the following order:

1. From JAR files located in OEBPS_HOME\ebmsapps\<application name>\lib
2. class files located in OEBPS_HOME\ebmsapps\common\classes
3. From JAR files located in OEBPS_HOME\ebmsapps\common\lib
4. class files located in OEBPS_HOME\ebmsapps
5. class files located in OEBPS_HOME\ebmsapps\common\ejbadapters

If there are multiple JAR files in any location, then BP Server sorts them in the ascending alphabetical order of their names and then searches in the same order.

If a class exists at more than one location, then it is loaded from the first location in which it is found as per the above order.

**Controlling dynamic class loader**

Dynamic class loading can be enabled or disabled by two methods:

- Setting the following value for the property in bpserver.conf
  
  bpserver.classloader.dynamic=true/false

  If this property is changed at runtime, after the server has been started, then "reload" needs to be called for BP Server so that the new property value is loaded.
• Using the command in BP Server Admin

\texttt{setDynamicCL \textit{true/false}}

The current value of dynamic class loading can be queried using the following BP Server Admin command:

\texttt{isDynamicCL}

BP Server considers only those JAR files for loading the classes that exist at the time of server startup. If new JAR files are placed after server startup, then the class loader needs to be refreshed so that it can pick the newly added JAR files. This feature is very useful to place the JAR files dynamically at runtime.

The refresh can be carried out by the following commands in BP Server Admin:

• \texttt{refreshClassLoader <ptName>}

This command reloads the JAR files from all the common paths as well as the process template specific path, \texttt{OEBPS_HOME\ebmsapps\<ptName>\lib}, where \texttt{<ptName>} is the process template name.

• \texttt{refreshClassLoader}

This command reloads the JAR files from all the common paths as well as the process template specific path \texttt{OEBPS_HOME\ebmsapps\<ptName>\lib} (where \texttt{<ptName>} is the process template name), for all the process templates whose adapters are loaded.

\section*{Modifying the JAR files}

When a JAR file is loaded by BP Server, some operating systems like Windows do not allow updating that JAR file. However, you can update the JAR file as follows:

1. Copy the JAR file to a new location.
2. Update the JAR file in the new location.
3. Copy the JAR file back to the previous location.

For example, to update \texttt{OEBPS_HOME\ebmsapps\common\lib\common_class.jar},

• Copy \texttt{OEBPS_HOME\ebmsapps\common\lib\common_class.jar} to \texttt{OEBPS_HOME\tmp} folder.
• Execute the command: \texttt{jar –uvf OEBPS_HOME\tmp\common_class.jar NewClass.class}
• Copy \texttt{OEBPS_HOME\tmp\common_class.jar} back to \texttt{OEBPS_HOME\ebmsapps\common\lib} folder.

\section*{Loading adapter classes}

Adapter classes can be loaded either by Application class loader or by Adapter class loader.

\section*{Application class loader}

Application Class Loader is the class loader, into which all the classes of an application (process template) are loaded by a single instance of the class loader. By default BP Server is enabled to use application class loader.
The benefit of Application Class Loader is that all the adapters related to a particular application are loaded by a single class loader. However, if one adapter needs to be reloaded, then all the previously loaded adapters also become invalid as a new instance of class loader needs to be created.

**Adapter class loader**

In Adapter class loader, each adapter is loaded by a different instance of the class loader. To enable Adapter class loader, set the following property in `bpserver.conf`:

```
bpserver.classloader.application=false
```

This property defines whether a different classloader is to be used for different applications. For any change in this property to be effective, BP Server must be stopped and restarted.

The benefit of Adapter class loader is that reloading an adapter does not affect any other loaded adapters. However, using Adapter class loader entails that there are as many class loaders as the adapters loaded in the system.

In production environments, adapter modifications are less common, and hence the dynamic class loading option can be disabled to improve the performance. We recommend you to start the BP Server with the following property value in the file `bpserver.conf`:

```
bpserver.classloader.dynamic=false
```

If new JAR files are added to the folder `OEBPS_HOME\ebmsapps\<application name>\lib` after BP Server startup, then those JAR files are automatically picked by the class loader to search for class files. This is true for application class loaders as well as Adapter class loaders.

If new JAR files are added to the common JAR file locations after BP Server startup, then they are picked by the class loader to search for class files whenever a new application attempts to load an adapter. Thereafter, the newly added JAR files are available to all the running applications.

**Dynamic class loader APIs**

Business Process Server provides two new APIs related to ClassLoader mechanism:

- `public void setDynamicClassLoading(Session session, boolean value)`
  
  Enables dynamic class loading.

- `public boolean isDynamicClassLoading(Session session)`
  
  Checks if dynamic class loading is enabled.

These APIs provide support for set/get status on Dynamic Class loading.
Process template versioning

In a business scenario, any installed application may need some modification at some stage. In such a case, the administrator may have to remove the existing process template and install the modified process template. This activity continues if the process template needs more revisions, and the version maintenance becomes difficult. Further, users have only the latest version available, and if they need an intermediate version, then the removal and addition may cause confusion.

BP Server provides the Process Template Versioning feature to overcome this problem by simplifying the version availability and maintenance.

Process Template Versioning allows the installation of a new version of an existing installed application.

**Note:** In Business Process Server, an application is an installed, executable business process, while a process template is a model of business flow which includes worksteps, connectors and dataslots. Once a process template is published and installed as an Business Process Server application, users can work on the new application.

Versioning enables you to install various versions of a template simultaneously, so that at any specific time, a single version remains activated and all other versions are deprecated. You can publish new versions of the installed application without removing the active application and all of its active instances, and also without stopping the BP Server.

Versioning is helpful when your existing process template is deprecated and the modified process template is installed with the same AppName. If a similar change is required again, then you can install a new version, and deprecate all the other versions. Whenever you need to work on any previously deprecated version, you must uninstall the versions created after that. BP Server automatically picks up the last version as the current one.

For details, see the following topics:

- Naming process templates
Chapter 9: Process template versioning

- Versioning and APIs
- Versioning example
- Creating process templates
- Accessing versions

Naming process templates

Business Process Server supports sequential versioning, where the different versions of a process template share the same Application Name, which is the name of the installed process template. Therefore, while versioning, you should follow this convention of naming the process templates:

- All process templates have a common application name (AppName) defined or inherited. Typically, it indicates the objective of the application.
- Each version of the process template is named as (AppName)_v (version ID).

For example, if theAppName is "Approval", then the first version should be named "Approval_v1", and the third version should be named "Approval_v3". These names are displayed in Progress Developer Studio for OpenEdge, and Business Process Portal.

Versioning and APIs

BP Server provides object-oriented APIs to support the creation, installation, and management of new versions for a process template.

Application versioning

With application versioning, there are no compatibility issues with applications from previous Business Process Server installations. To take advantage of application versioning, customized portals work with the BP Server API to retrieve all active applications in the BP Server. BP Server API and BP Server Admin support the creation and installation of new versions of an application.

In application versioning, two versions of an application are installed using two different process templates, usingAppName to link the various versions of the same application installed with different process templates.

When the first version of an application is installed,AppName is set to the process template name by default. Subsequent application versions require installation with the API:

- createAndInstallProcessTemplateVersion(String session, String xmlFileName, String parentPTName)

When the new version is published installing this API, it deprecates the parent process template and sets theAppName of the new template to that of its parent. We recommend not changing theAppName while using the API createAndInstallProcessTemplateVersion to install different versions.

The Table 6: BP Server APIs for application versioning on page 67 provides a summary of the BP Server APIs for application versioning.
Table 6: BP Server APIs for application versioning

<table>
<thead>
<tr>
<th>API name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>createAndInstallProcessTemplateVersion</td>
<td>Creates and installs a given process template.</td>
</tr>
<tr>
<td>getAppName</td>
<td>Returns the appName for a given process template.</td>
</tr>
<tr>
<td>getUserProcessTemplateList</td>
<td>Returns a list of all authorized process templates for the specified user in a hashtable.</td>
</tr>
<tr>
<td>getProcessTemplateVersions</td>
<td>Returns all related installed versions of an application for the specified appName. This API assumes that all versions of the same application use the same appName.</td>
</tr>
</tbody>
</table>

Versioning example

Consider a scenario where an administrator has used ACME_Process application to create ACME_Process_Ver1, which is active in the BP Server.

Suppose that the organization policy is changed, and the ACME_Process_Ver1 application needs changes. The application developer can modify the ACME_Process_Ver1 process template and create a new process template called ACME_Process_Ver2.

However, there are some active instances of the application ACME_Process_Ver1 and the application developer wants to retain version 1. Using the process template versioning, the application developer wants to install a new version of application with the same appName.

When the new version ACME_Process_Ver2 is installed, it becomes the active version, and the previous version ACME_Process_Ver1 becomes deprecated. No user can create new instances of ACME_Process_Ver1, as the template is deprecated.

Versioning allows the parent process template to have a single child (version), maintaining a one-to-one relationship between parent and child. In this example, when ACME_Process_Ver2 is installed, it is the child of ACME_Process_Ver1. In future, ACME_Process_Ver2 may also be modified to create ACME_Process_Ver3. In that case, ACME_Process_Ver3 will be the child of parent ACME_Process_Ver2.

Figure 6: Parent-child relationship in versions on page 67 illustrates the parent-child relationships.

Note that ACME_Process_Ver3 is not a child of ACME_Process_Ver1.
Creating process templates

In process template versioning, multiple versions of an application are installed using different process templates. During this time, the *AppName* is set to the process template name by default. Initially, the user must create and install the process template, and then the subsequent application versions can be installed with help of `createVersion()` API. This deprecates the parent process template and sets the *AppName* of the new template to that of its parent.

Installing process template

You can create and install the ProcessTemplate with the help of the following BLServer `createProcessTemplate` API.

- `public createProcessTemplate(Session, String, boolean)`

Creating versions

When the requirements change, you may need to create a version *ACME_Process_Ver2* for the existing process template, without removing *ACME_Process_Ver1*. This can be accomplished with the help of API provided by the ProcessTemplate SVO.

- `public ProcessTemplate createVersion(String xmlFileName)`

The following scenarios explain the versioning in more detail:

- You can remove the parent process template, which is deprecated, only when all its existing process instances are completed.
- If you change the *AppName* of the latest active version, then the new *AppName* is reflected on all the versions.
- If you remove the latest (active) version, then you can resume any of the deprecated versions.

**Note:** Versioning does not allow you to activate multiple versions simultaneously.

Accessing versions

BP Server provides the following two client-side Smart Value Objects (SVOs) to access and manage the process templates.

- ProcessTemplate
- ProcessTemplateList

Getting version list

BP Server provides two APIs by which you can get the process template version list.

- `public ProcessTemplateList getProcessTemplateVersions(Session, String)`
This API is of class BLServer. All the versions of the ProcessTemplate SVOs can be obtained with the given
AppName.

• public ProcessTemplateList getVersions()

This API is of class ProcessTemplate and retrieves the list similar to getProcessTemplateVersions() API.

Getting version ID

ProcessTemplate provides the following API to get the ID of the version.

• public String getVersionID()

The API is of class ProcessTemplate and is used to get the Version ID of the process template. Version ID is assigned to ProcessTemplate at the time of creating the application in Progress Developer Studio for OpenEdge.

If the application developer does not assign any value to Version ID at the time of creating a process template, then by default it is taken as 1.0.

Version ID is used only for informing the user, and can be changed after creating, installing, and running the application.

Changing Version ID requires restarting the BP Server, and reinstalling the application.

Getting version state

ProcessTemplate provides the public boolean isSuspendedBySeqVersion() API to get the state of the version.

Check whether this process is suspended by the sequential versioning. Whenever a newer version of an application is installed, all the existing versions of the ProcessTemplate and their instances are suspended. Using this API, the user can verify whether the process is suspended.
Workstep performers

A performer is an entity that executes a workstep. This chapter describes different types of performers, their interdependencies with each other, and how you can exclude a particular performer.

For details, see the following topics:

• Types of performers
• Understanding workstep performer hierarchy
• Excluding performers for worksteps

Types of performers

BP Server allows you to specify following types of performers:

• Single user
• Group
• Queue
• External adapter

Except the adapter, all these are human performers, and are used for human worksteps, whereas the adapter is used for adapter worksteps.
If an e-mail ID is specified as a performer for a human activity workstep, then an e-mail is sent to that e-mail ID on activation of that workstep. This e-mail contains input/output dataslot values. The user can complete this task by replying to this e-mail and providing values for output dataslots in that reply. On receipt of this e-mail, BP Server completes the task along with the output dataslot values.

**Note:** Make sure the e-mail ID you specify as an e-mail performer in the Progress Developer Studio for OpenEdge is valid. No action is taken, and no errors are logged, if the e-mail address is invalid.

If a workstep with a queue performer is configured as an "e-mail performer", then to send e-mails to all members of the queue, ensure that the Group implementation of the Realm implements com.tdiinc.userManager.AdvanceGroup interface. Both com.tdiinc.userManager.JDBCRealm and com.tdiinc.userManager.LDAPRealm support AdvanceGroup.

---

**Specifying performer precedence**

You can specify a user, a user group, or a queue with the same name. In such cases, by default, BP Server evaluates the performers in the following order:

- User
- Group
- Queue

You can reverse this order by setting the `bpserver.resource.validation.ascendings` parameter to "false" in the OEBPS_HOME\conf\bpserver.conf file.

---

**Understanding workstep performer hierarchy**

For all Activity worksteps, Business Process Server checks the validity of the workstep performer. After this check is performed, the task is assigned (or made available) to the user. For non-existent or unavailable performers, Business Process Server manages tasks, by following the performer hierarchy described in Table 7: Workstep performer hierarchy on page 72.

**Table 7: Workstep performer hierarchy**

<table>
<thead>
<tr>
<th>Workstep performer</th>
<th>If invalid, then ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original task performer</td>
<td>Business Process Server checks the validity of a default performer, defined in Progress Developer Studio for OpenEdge. For more information, refer to the OpenEdge Getting Started: Developing BPM Applications with Developer Studio. If the default performer is a valid user, then the task is assigned (or made available) to that user.</td>
</tr>
</tbody>
</table>
If invalid, then...

<table>
<thead>
<tr>
<th>Workstep performer</th>
<th>If invalid, then ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default Performer (Manager)</td>
<td>Business Process Server checks the validity of a back-up performer, which could be a single user or a user group, specified in the Process Properties dialog box in Progress Developer Studio for OpenEdge. For information on specifying the back-up performer, refer to the “Defining Process Properties” section of the OpenEdge Getting Started: Developing BPM Applications with Developer Studio. If the back-up performer is a valid user, then the task is assigned (or made available) to that user.</td>
</tr>
<tr>
<td>Back-up Performer</td>
<td>The workstep is suspended. Any user with managerial privileges can then resume the workstep.</td>
</tr>
</tbody>
</table>

**Default performer for tasks assigned to user groups**

A task can be assigned to a Business Process Server group, either with the **All** option or with the **Any** option.

**Group task options**

For group tasks with **All** option, if any of the group members is invalid, then the task is assigned to the valid default performer. In this case, the default performer needs to be a single user.

For group tasks with **Any** option, if all the group members are invalid, then the task is made available to the valid default performer. In this case, the default performer can either be a single user or a group.

If an invalid performer is specified for a task assigned to a group with **All** option, then the task is assigned to the manager. If the manager is a single user, then the task is assigned to him/her; if it is a group, then the task is assigned to the first valid member of the group.

If an invalid performer is specified for a task assigned to a group with **Any** option, then the task is made available to the manager. If the manager is a single user, then the task is assigned to him/her; if it is a group, then the task is made available to the group.

If the manager is an invalid user, then the task is suspended.

**Back-up performer for group assigned tasks**

As described in Table 7: Workstep performer hierarchy on page 72, a task is assigned (or made available) to the back-up performer, only if the default performer is invalid. Further:

- For group tasks with **All** option, if any of the group members is invalid, then the task is assigned to the valid back-up performer.
- For group tasks with **Any** option, if all the group members are invalid, then the task is made available to the valid back-up performer.

**Excluding performers for worksteps**

For major business decisions, especially in the finance domain, it is important that more than one decision-maker is involved. Known as the Four-eye principle, this approach ensures that in the collaborative environment, at least two persons review and approve any significant process. For example, analysts and examiners are jointly in charge of surveillance of banks or financial institutions, establishing that the supervisory activities of any bank are governed by the Four-eye principle.
The same principle is considered as a good practice for design and development, and in review, guaranteeing good quality of the products. For example the reviewer of a workstep is excluded from reviewing another task that reviews the same thing.

The Four-eye principle in business processes means exclude the performer of one workstep from being a performer of another workstep. For example, if a manager creates a purchase request, then the same manager must not approve that purchase request. In other words, the manager is "excluded" from the "approve" workstep, and some other performer must be specified. It is especially important when the performer may be a group, queue, or even an individual user.

BP Server supports the Four-eye principle by implementing the exclude performer list for worksteps, where the performers of one or more worksteps may be excluded for a workstep.

### Specifying the exclude performers

The format of the ExcludePerformers list is:

```xml
<ExcludePerformers>
  <Performer name="admin" />
  <Performer name="@ds1" />
  <PerformerOf name="Activity 2"/>
</ExcludePerformers>
```

- The performer can be either a valid performer or a mapped dataslot. More than one <Performer> tag can be used to specify multiple exclude performers. However, each tag can have only one user or mapped dataslot value.
- `<PerformerOf>` tag can be used to specify the workstep name whose performer should be excluded. You must specify the workstep name explicitly.

Note: Mapped Dataslots cannot be specified for the `<PerformerOf>` tag.

In general, for each user, workstep, and EJB application, a separate tag should be specified. The mapped dataslots and EJB applications can return only individual users.

Note: The excluded performer list should not include groups and queues.

Examples of the Exclude Performer format are given in the Workstep properties in OpenEdge on page 75.

The BP Server excludes or removes each resource in the exclude list from the performer list. Any resource in the exclude list that is not available in performers list of a workstep is ignored. After the exclusion of all specified performers, the work item is either Assigned or Available to the final list of performers.

A user can complete a task delegated to him/her even if he/she is excluded from the performer list.

### Actual performer of completed work item

A successfully completed work item is always removed from the BP Server database cache. Consequently, the actual performer of a completed work item is not recorded. In practice, a work item may be reassigned to another person who is not even specified in the workstep performer list, and who may complete it. It is therefore necessary that the real performer who has completed the work item should be recorded for future use in the business process.

The excludeWSList is the list of workstep names that were specified as the value of the `<PerformerOf>` tag.
For any work item completion, BP Server verifies in the process template whether the *excludeWSList* is empty. If this list is empty, then BP Server continues as before.

BP Server writes the performer of the completed work item in the database only when the workstep name of the completed work item exists in the *excludeWSList*. In this approach, any overhead to applications that does not use the Four-eye principle is limited to one in-memory operation.

If a process template has worksteps with exclude-performers lists, then the records inserted in the above table are removed only on completion or removal of a process instance.

### Workstep properties in OpenEdge

Atomic workstep properties include one additional property, *Exclude Performer List*.

The following examples show how the `<ExcludePerformers>` tag can be used to write this value in process template XML.

- **Performer User**

  ```xml
  <AtomicWS name="Approve" Name="Approve">
  <Performer>queue1</Performer>
  <ExcludePerformers>
  <Performer name="user1"/>
  <Performer name="user3"/>
  <PerformerOf name="ws1"/>
  </ExcludePerformers>
  </AtomicWS>
  ```

- **Performer of Workstep**

  ```xml
  <AtomicWS name="Approve" Name="Approve">
  <Performer>queue1</Performer>
  <ExcludePerformers>
  <Performer name="user1"/>
  <Performer name="@ds1"/>
  <PerformerOf name="ws1"/>
  <PerformerOf name="ws3"/>
  </ExcludePerformers>
  </AtomicWS>
  ```

- **Performer EJB Applications**

  ```xml
  <AtomicWS name="Approve" Name="Approve">
  <Performer>queue1</Performer>
  <ExcludePerformers>
  <Performer name="jndi://$default/TestEJB?getPerformers"/>
  </ExcludePerformers>
  </AtomicWS>
  ```

Alternatively, each exclude performer can also be written with a separate `<ExcludePerformer>` tag value. The performer list contains only comma-separated values.
Dynamic user profile

Business Process Server supports the following user management realms:

- JDBC
- LDAP
- LDAP Hybrid

For LDAP, and LDAP Hybrid realms, Sun Java System Directory, and Microsoft Active Directory are supported. However, it is possible to replace the default realm of Business Process Server with a custom implementation. For this purpose, Business Process Server supports pluggable realm implementations by providing standard interface to all the realms. You can implement the Business Process Server user management interface to replace the default Business Process Server implementation code.

However, some applications — where users, groups, roles, and queues are maintained and managed by external applications (such as client-developed custom portals) — may in turn call the BP Server. In such cases, BP Server supports specifying user profile information at runtime using the BP Server session object.

For details, see the following topics:

- Working with the UserProfile object
- Using the UserProfile object with queues

Working with the UserProfile object

The BP Server Session object supports the following services to specify the user profile for a session user at runtime.
public void setUserProfile (UserProfile userProfile)

public UserProfile getUserProfile()

The UserProfile object contains attributes which are saved as name-value pairs in a Map. These attributes must be defined and their values must be provided by the external application. BP Server provides the following methods to access attributes through a Session object:

• public void setUserAttribute(String key, Object value)
• public Object getUserAttribute(String key)

Any change to the UserProfile object that is already associated with a session is effective only when one of the server side methods is invoked.

Using the UserProfile object with queues

When the findQueues value is set to false on UserProfile object by calling the setFindQueues() method, BP Server uses only the queues specified on the UserProfile object while getting the work item list available to the user. But if queues specified on UserProfile objects are null, then BP Server gets a list of all queues that the user belongs to, and also a list of all queues that contain any of the groups that the user belongs to. It then sets the queue list on UserProfile object and uses it for getting available work item list for the user.

When the findQueues value is set to true on UserProfile object by calling the setFindQueues() method with parameter value as 'true', BP Server first finds all groups that the user belongs to. Then it uses the user and all the group names to find all the queues where the user or the groups are members. The retrieved queue lists are added to the already specified queue list in UserProfile, and this is used for getting the available work item list for the user.

Note that queues are used only when the oebps.queues is set to true in the OEBPS_HOME\conf\oebps.conf file.

The following method on com.savvion.sbm.bizlogic.server.svo.WorkItemList class is one of the ways you can fetch available work items for the session user:

public static WorkItemList getAvailableList(Session session)
Query service

The BP Server is the workflow engine that processes user requests. Most of the time, users request to retrieve bulk data from the server. The data retrieved from the server needs to be serialized on the server and deserialized on the client, and that effects the performance. Usually users apply filtering and specify the order of the data to be retrieved. For example, a particular user may be interested to see only the tasks that are due in the next two days, instead of all the assigned tasks.

Query Service is a BP Server client utility for faster retrieval of workflow information. It is a part of the BP Server client library that can run in the client workspace to retrieve the read-only data, based on the user queries. Query Service is an effective utility to reduce the network traffic and the load on the BP Server.

For details, see the following topics:

• Working with queries
• Working with filters
• Query service results
• Configuring Business Process Server for query service

Working with queries

Query Service is the entry point for the BP Server client library for retrieving bulk data related to user's tasks and process instances created by users. The library has to be initialized once for each JVM, and cleaned up before the shutdown of the JVM.

You can add filtering conditions to the query to reduce the amount of data to be retrieved. Query Service can optionally pass filters that contain:
- Additional conditions
- Sort order of the retrieved data
- Additional column containing dataslots to be retrieved
- The size of the data to be retrieved
  
  If the data size is not specified, then it is equal to the size of the retrieved ResultSet.

**Note:** BP Server client library is a read-only library and should not be used for updating the workflow data. You can use the BP Server for updating the workflow, process template, metadata, and dataslot metadata.

BP Server Smart Value Objects (SVO) for WorkItem, WorkStepInstance and ProcessInstance can be retrieved directly from the QSResult.

*Figure 7: Query service client library* on page 80 presents the client library structure.

*Figure 7: Query service client library*

```plaintext
QGFilter

QGProcessInstanceFilter  QGWorkItemFilter  QGWorkStepInstanceFilter

QGResult

QGProcessInstanceRS  QGWorkItemRS  QGWorkStepInstanceRS

QGProcessInstanceData  QGWorkItemData  QGWorkStepInstanceData
```

Every user obtains the *QueryService* handle by passing a valid BP Server session object. The BP Server session object is necessary to communicate with the BP Server using the SVO obtained from the *QueryService*. When a user gets a new session, `setSession` is invoked.

The *Table 8: Query service classes* on page 80 describes the classes provided by Query Service.

*Table 8: Query service classes*

<table>
<thead>
<tr>
<th>Class</th>
<th>Returns</th>
<th>Retrieves data for ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>QSWorkItem</td>
<td>QSWorkItemRS</td>
<td>Work item</td>
</tr>
</tbody>
</table>
Working with filters

<table>
<thead>
<tr>
<th>Class</th>
<th>Returns</th>
<th>Retrives data for ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>QSWorkStepInstance</td>
<td>QSWorkStepInstanceRS</td>
<td>WorkStepInstance</td>
</tr>
<tr>
<td>QSProcessInstance</td>
<td>QSProcessInstanceRS</td>
<td>ProcessInstance</td>
</tr>
</tbody>
</table>

**QSWorkItem**

*QSWorkItem* provides data for work items which are assigned and available to a user, including the work items delegated to them by the other users. Every request can optionally take a *QSWorkItemFilter*. The return type is a *QSWorkItemRS* that provides the scrollable ResultSet and the WorkItem SVO. The default sort order for the retrieved data is due date.

The APIs used with the delegated work items are called proxy APIs, which get available and assigned work items using query service support queues.

**QSWorkStepInstance**

*QSWorkStepInstance* provides the WorkStepInstance data for the ProcessInstance created by the user. Every request can optionally take a *QSWorkStepInstanceFilter*. The status of the retrieved WorkStepInstance is Activated or Suspended. Any WorkStepInstance that is created and completed cannot be retrieved. The default sort order for the retrieved data is the status of the WorkStepInstance.

**QSProcessInstance**

*QSProcessInstance* provides the WorkStepInstance data for the ProcessInstance created by the user. Every request can optionally take a *QSProcessInstanceFilter*. The status of the retrieved ProcessInstance is Activated or Suspended. Created, Completed, and Removed WorkStepInstance cannot be retrieved. The default sort order for the retrieved data is the status of the ProcessInstance.

**Working with filters**

Often, a user may have a large number of tasks that have a wide range of dates for due date. Quite often, only the tasks due in the very near future may need the user’s immediate attention. In some other cases, a user may focus on WorkStepInstance or ProcessInstance while retrieving data.

BP Server Query Service supports the following types of filters to focus on relevant data while retrieving:

- QSWorkItemFilter for WorkItem requests
- QSProcessInstanceFilter for ProcessInstance requests
- QSWorkStepInstanceFilter for WorkStepInstance requests

A Filter can also be applied to a specific application.

The Table 9: BP Server database tables on page 82 lists database table names of BP Server, available with their table alias which can be referred in the Filter.
### Table 9: BP Server database tables

<table>
<thead>
<tr>
<th>Table name</th>
<th>Table alias name</th>
<th>Contains ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIZLOGIC_WORKITEM</td>
<td>BLWI</td>
<td>Work item information</td>
</tr>
<tr>
<td>BIZLOGIC_AVAILABLEWORKITEM</td>
<td>BLAWI</td>
<td>Information about work items that are available to performers</td>
</tr>
<tr>
<td>BIZLOGIC_PROCESSINSTANCE</td>
<td>BLPI</td>
<td>ProcessInstance information</td>
</tr>
<tr>
<td>BIZLOGIC_WORKSTEPINSTANCE</td>
<td>BLWSI</td>
<td>WorkInstance information</td>
</tr>
<tr>
<td>BIZLOGIC_DS_ptid(^2)</td>
<td>BLIDS</td>
<td>Instance dataslot table</td>
</tr>
<tr>
<td>BIZLOGIC_GLOBALDS_ptid(^2)</td>
<td>BLGDS</td>
<td>Global dataslot table</td>
</tr>
</tbody>
</table>

When the dataslots are mentioned in the filter, the retrieved list is only for that process even if the process has versions. If the dataslots are not mentioned, then the retrieved list is across the versions of the process. The column name in the filter must be in uppercase. When additional columns are mentioned, they can only be of dataslot names, and should always be prefixed either with BLIDS or BLGDS followed by the dataslot name.

The status of the Workitem, WorkStepInstance, and ProcessInstance are stored as numbers. QSConstants class has integer constants defined for the status.

The Table 10: Process states as QSConstants on page 82 lists the process states as QSConstants.

### Table 10: Process states as QSConstants

<table>
<thead>
<tr>
<th>Process State</th>
<th>QSConstants</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active ProcessInstance</td>
<td>QSConstants.PI_ACTIVATED</td>
</tr>
<tr>
<td>Suspended ProcessInstance</td>
<td>QSConstants.PI_SUSPENDED</td>
</tr>
<tr>
<td>Active WorkStepInstance</td>
<td>QSConstants.W_ACTIVATED</td>
</tr>
<tr>
<td>Suspended WorkStepInstance</td>
<td>QSConstants.W_SUSPENDED</td>
</tr>
<tr>
<td>Wait WorkStepInstance</td>
<td>QSConstants.W_WAIT</td>
</tr>
<tr>
<td>Available Work item</td>
<td>QSConstants.I_AVAILABLE</td>
</tr>
<tr>
<td>Suspended Work item</td>
<td>QSConstants.I_SUSPENDED</td>
</tr>
<tr>
<td>Assigned Work item</td>
<td>QSConstants.I_ASSIGNED</td>
</tr>
</tbody>
</table>

### Filter properties

A filter has the following properties:

- Name — Name of the Filter.

\(^2\) ptid is the process template ID for the process they represent, and is different for different applications.
• Type — WorkItem, WorkStepInstance, or ProcessInstance type.

• Condition — Condition can be set on the filter. QSWorkItemFilter condition should not contain the performer of the WorkItem, and the WorkStepInstance and the ProcessInstance Filter condition should not contain the creator of the process instance.

• Additional columns — It might be necessary to retrieve dataslot values with WorkStepInstance data. Therefore, additional columns can be added to the Filter in java.util.List. Note that you can use only dataslots in the additional column.

• Order By — The order by which the data is to be retrieved can be specified in the Filter.

• ProcessName — The application name can be optional in the Filter. However, if a dataslot is used in the condition, additional columns, or order properties, then the process name is mandatory.

• FetchSize — The size of the SVO for every request to the Filter may be more than 500 items, but the user may want only the first 100 items at any time. In that case, the fetch size can be set to 100 items. The returned SVO list size is equal to 100 items or the ResultSet size, whichever is less. If the user does not set fetch size explicitly, then the default value is 100 items. User can set the fetch size by calling the setFetchSize() method on the QSFilter object.

• Connection — It is required sometimes for the QueryService requests to be part of an existing client transaction. In that case, a connection object can be set in the Filter. Note that the Connection is a transient attribute in the Filter, which means the Filter object loses the reference of the Connection during serialization and deserialization.

**Constructing Filters**

Keep in mind the following guidelines while constructing a filter.

• When any of the columns are mentioned in the filter, the table alias must be prefixed with them.

• When the dataslot names are mentioned in the columns, they must be in uppercase and must also be enclosed in double quotes since the dataslot name can contain space.

• When any of the time related columns like STARTTIME, ENDTIME, and DUEDATE are used in the Filter, QSClientUtil.getFilterTime(long time in msec) needs to be invoked.

For example, if any of the work item due date is past due, then the condition has to be:

```java
long currentTime = System.currentTimeMillis();
long duedate_value = QSClientUtil.getFilterTime(currentTime);
String condition = "BLWI.DUEDATE <= " + duedate_value;
QSWorkItemFilter wifil = new QSWorkItemFilter("wi_currentdue",null);
wifil.setCondition(condition);
```

• The priority is stored as NUMBER in the database.

For example, if the priority value is high, then the value stored in the database may be 30. So it is recommended to use QSClientUtil.getIntegerPriority(String str_priority) whenever the PRIORITY is used in the condition.

• Any additional column set in the Filter must have the dataslot table alias prefixed with the dataslot name.

• LIST dataslots are stored as BLOB in the BP Server dataslot table along with other object dataslots under the column called LARGE_DATASLOTS. When the LIST dataslots are to be retrieved using QueryService, the additional column must have an Alias with the LIST Dataslot name in uppercase. For example, if an
application has a LIST dataslot by name *lstDS* of instance scope, then to add the LIST dataslot in the additional column, the column name should be BLIDS.LARGE_DATASLOT AS LSTDS.

- LIST and OBJECT dataslots cannot be used in the Condition and Order by properties of the Filter.

**Filter constraints**

For the supported filters, the following constraints apply:

- QSWorkStepInstanceFilter should not contain:
  - any columns from BIZLOGIC_WORKITEM table.
  - CREATOR column of BIZLOGIC_PROCESSINSTANCE table in the Condition attribute.

- QSProcessInstanceFilter should not contain:
  - any columns from BIZLOGIC_WORKSTEPINSTANCE and BIZLOGIC_WORKITEM table.
  - CREATOR column of BIZLOGIC_PROCESSINSTANCE table in the Condition attribute.

- QSWorkItemFilter should not contain:
  - PERFORMER column of BIZLOGIC_WORKITEM table in the Condition attribute.
  - AVAILABLEFOR of BIZLOGIC_AVAILABLEWORKITEM in the Condition attribute.

**Parameterized condition support for filter**

BP Server supports parameterized condition for filter. This allows you to set a condition containing 'Named Parameters'. After setting the condition once, you must call the `setParameterValues(Map params)` method and pass values for all named parameters. This allows you to reuse the filter object. You can create a filter object and set conditions on it. You can then save this filter object in a persistence storage for reuse. From that time on, while using this filter object to fetch data, you only need to set the parameters on it. Even though parameters are passed as a map, internally they are put into a query string in appropriate order.

For example, you can use named parameters in a WorkItem filter as shown below.

```java
QSWorkItemFilter fil = new QSWorkItemFilter("wifil");
HashMap<String, Object> params = new HashMap<String, Object>();
params.put("longds", 20);
params.put("strds", "banana");
fil.setCondition("BLIDS.STRDS = #strds# and BLIDS.LNGDS= #longds#); 
fil.setParameterValues(params);
```

**Query service results**

Every request to the QueryService library returns QSResult that wraps the BP Server SVO. There are three types of QSResult:

- **QSWorkItemRS** — WorkItem requests.
- **QSWorkStepInstanceRS** — WorkStepInstance requests.
• **QSProcessInstanceRS** — ProcessInstance requests.

The type of SVO returned depends on the type of QSResult as described in Table 11: QSResult and SVOs on page 85.

### Table 11: QSResult and SVOs

<table>
<thead>
<tr>
<th>QSResult</th>
<th>SVO type</th>
</tr>
</thead>
<tbody>
<tr>
<td>QSWorkItemRS</td>
<td>QSWorkItemData</td>
</tr>
<tr>
<td>QSWorkStepInstanceRS</td>
<td>QSWorkStepInstanceData</td>
</tr>
<tr>
<td>QSProcessInstanceRS</td>
<td>QSProcessInstanceData</td>
</tr>
</tbody>
</table>

The size of SVO can be controlled from the QSResult, for the full list, a particular index, or a range of indices.

- **QSResult.getSVOList()** returns the complete list, and empty `java.util.Vector` if the size is 0.
- **QSResult.getSVOList(from, to)** returns a portion of the list between the specified `from Index and to Index`. The returned list includes the `from Index`, but excludes the `to Index`. If the `from Index` and the `to Index` are the same, then the returned list is empty. The returned list is backed by the complete list, so that changes in the returned list are reflected in the complete list, and vice versa. The returned list supports all of the optional list operations supported by the complete list. If the `to Index` is greater than the actual size of the complete list, then the `to Index` is equal to the size of the complete retrieved list.

It throws an exception for an illegal endpoint index value, such as `from Index` being less than or equal to `Index`. Refer to `java.util.List` for more details.

- **QSResult.getSVO(index)** returns the SVO if the index is valid. Otherwise, an exception is raised. Note that the exception is raised if the index is out of range.

**Note:** The `getSVOList` returns a list of SVOs. The value of each Index must be cast to the appropriate SVO type before any operation is requested. Similarly, `getSVO` returns BLProcess but must be cast to the appropriate SVO type before any operation is requested.

The index passed should follow the Java array syntax, where the index starts from 0 and not 1. If the size is 10, then the last index is 9.

If additional columns are specified in the filter, then `getAdditionalColumn()` returns a HashMap, with keys as column names (alias names if specified), and values as:

- **OBJECT**, if the dataslot type is object / collection type. The object can be retrieved based on the Alias mentioned, or on the collection dataslot in the column definition of the Filter, and can be cast into the appropriate object type.
- **String** for all other types.
  - For **LOGICAL** dataslots, the value is 1 for true and 0 for false.
    ```java
    String boolValue = (String)hm.get("BOOLDS");
    boolean bool = (Boolean.valueOf(boolValue)).booleanValue();
    ```
  - For **INTEGER** dataslots, the Decimal SVO can be constructed as follows:
    ```java
    String decimalValue = (String)hm.get("DECIMALDS");
    Decimal decimal = new Decimal(decimalValue);
    ```
• The DATETIME-TZ SVO can be constructed as follows:

    String datevalue = (String)hm.get("DATETIME-TZ");
    DateTime dt = new DateTime(datevalue);

**Note:** A valid BP Server session should be passed if the SVO obtained has to communicate with BP Server.

### Configuring Business Process Server for query service

To configure Business Process Server for query service, oebpsclient.conf must be in the class path that contains the following configuration information.

• **qs.use.datasource**

  The value should be either "true" or "false". The default value is "true". The DataSource must be in the same JVM on which the QueryService is running. If the DataSource is in another JVM, then performance is affected.

  If the DataSource is not available, then the ConnectionPool is used. If the ConnectionPool is not available, then an error is raised.

• **qs.cpool.minConn**

  This value should be an integer, the default value of minimum connection is 5. The configured connection should be at least 5, and should not exceed 30.

• **qs.cpool.maxConn**

  This value should be an integer. The default value of maximum connection is 30. The configured connection should be at least 30.

**Note:** Specifying a very high number of maximum connections will overload the connection resources.

The logging information for query service is provided by oebpslog.conf. A category for QueryService called **BP ServerClient** is also included.

If there is an error, com.Savvion.sbm.bizlogic.client.queryservice.QSException is raised.
Smart value objects

BP Server provides a number of Smart Value Objects (SVOs) which act as data transporters between the BP Server and the client. The SVOs improve performance and allow transparent communication among the BP Server components.

The structure and class diagram of SVOs are described in this chapter.

For details, see the following topics:

- Process SVOs
- Collection SVOs
- Dataslot SVOs
Process SVOs

The structure of Process SVOs is shown in Figure 8: Process SVO class diagram on page 88.

Figure 8: Process SVO class diagram

The class structure is described below:

• **BLProcess** — This abstract class is the super class of all SVOs and provides the common framework.

• **Business ProcessTemplate** — This is a client-side SVO to manage process template.

• **ProcessInstance** — A ProcessInstance object represents an active business process. It is a running instance of the ProcessTemplate.

• **WorkStepTemplate** — Worksteps are the basic building blocks of the workflow processes. WorkStepTemplate is the template level object of a workstep and provides the blueprint for a WorkStepInstance. For more information, see Workstep template management on page 111.

• **WorkStepInstance** — This SVO is actually executed during runtime. For more information, see Workstep Instance Management on page 131.

• **ActivityWSInstance** — This class extends WorkStepInstance and it is a SVO for WorkStepInstances performed by users.

• **WorkItem** — This is a client-side SVO to manage workitems. For more information, see Work item management on page 121.
Collection SVOs

The structure of Collection SVOs is shown in Figure 9: Collection SVO class diagram on page 89.

Figure 9: Collection SVO class diagram

The class structure is described below:

- **BLCollection** — This abstract class is the super class of all Collection SVOs and provides the common framework.
- **ProcessTemplateList** — This class provides methods for performing on list of ProcessTemplate SVOs.
- **ProcessInstanceList** — This SVO maintains a list of ProcessInstance objects.
- **WorkStepInstanceList** — This SVO maintains a list of WorkStepInstance objects. For more information, see Workstep Instance Management on page 131.
- **WorkItemList** — This SVO maintains a list of WorkItem objects. For more information, see Work item management on page 121.
- **DataSlotList** — This SVO maintains a list of DataSlot objects. For more information, see Datatot management on page 139.

Dataslot SVOs

The structure of Dataslot SVOs is shown in Figure 10: Dataslot SVO class diagram on page 89.

Figure 10: Dataslot SVO class diagram

- **AbstractDataSlot** — This is an abstract class for dataslots. Both DataSlot and DataSlotTemplate should be derived from this class. Except for the name and value of the DataSlot, all other attributes have been
taken as metadata. The metadata comprises of Type, Size, Multi line, Append with, BPM Manage API Access, Is public, Choices, Object type, Constructor data, Label, and IsRequired

- **DataSlot** — This SVO provides methods that can be performed on dataslots. For more information, see DataSlot management on page 139.

- **DataSlotTemplate** — SVO for dataslot attributes and metadata. For more information, see DataSlot management on page 139.

The Dataslot type interface, and its classes is shown in Figure 11: Dataslot type interface diagram on page 90.

**Figure 11: Dataslot type interface diagram**

- **DateTimeTZ** — This SVO provides methods that can be performed on DateTimeTZ type of dataslots.

- **Decimal** — This SVO provides methods that can be performed on Decimal type of dataslots.

For more information, see DataSlot management on page 139.
Application management

In Business Process Server, an application is an installed, executable business process that automates a business flow. Different versions of processes could be grouped together as one application, with a single application name. In other words, a BP Server application is a container that hosts multiple versions of a business process template.

Depending on the business requirement, one of the versions is installed as the current application. Newer versions can always be installed if the requirements change, and the older versions are internally deprecated.

For details, see the following topics:

- Accessing applications
- Using BP Server APIs
- Creating applications
- Retrieving applications
- Removing applications

Accessing applications

BP Server provides a client-side Smart Value Object (SVO) called "application" to manage the applications.
Using BP Server APIs

BP Server provides object-oriented APIs for application management. The client first needs to connect to the BP Server before executing any BP Server APIs.

Since Application, ProcessTemplate and ProcessInstance are all related SVOs, you can find several methods of ProcessTemplate or ProcessInstance can be run for the specified Application.

Creating applications

The first step in creating a new process is defining a new application name. This can be considered as creation of the first version of the Application. An Application SVO provides APIs for creating the first version as well as the future versions of the same Application.

- `Application.create(session, appName, xmlFileName);`
  This creates the first version of the Application.
  - If the `xmlFileName` does not contain an application name, then the passed `appName` is set as the application name.
  - If the `xmlFileName` contains an application name and the `AppName` passed is different from the application name, then an exception is raised.

- `Application.create(session, xmlFileName);`
- `Application.create(session, ByteArrayInputStream);`
  This creates the first version of the Application.
  - If the `xmlFileName` contains an application name, then it is used.
  - If the `xmlFileName` does not contain an application name, then the `ProcessTemplate` name is taken as the application name.

If the `xmlFile` is present on the client and not present in the location where Business Process Servers are running, then the `xmlFile` could be passed as an InputStream. A static method is provided on the Application SVO for creating the InputStream.

Once an Application is created, its future versions are created as TemplateVersion. For creating TemplateVersions, you can use the following APIs:

- `createTemplateVersion(xmlFileName);`
- `createTemplateVersion(ByteArrayInputStream);`

  The passed `xmlFileName` (ProcessTemplate XML File) is used for creating the current version of this Application.
  - If the `xmlFileName` contains an application name and it is different from the application name of the current application, then an Exception is raised.
  - If the `xmlFileName` does not contain an application name, then the application name of the current application is considered as the application name for the created version.
Retrieving applications

The following APIs can be used to retrieve the existing applications:

- `Application.getList(session);`
  This returns all the Applications currently available in the system as an ArrayList containing Application SVOs.

- `Application.get(session, appName);`
  This returns the Application SVO with the `AppName` which is passed on as its application name. If there is no application with the name passed, then an exception is raised.

- `Application.isExist(session, appName);`
  This returns "true" if an application with the name passed exists. If it does not exist, then it returns "false".

At any time there is only one ProcessTemplate active for an Application. All the previous versions of the ProcessTemplates are internally deprecated.

Removing applications

- `public void remove();`
  This removes the current Application.

- `public void removeAllInstances();`
  This removes all the ProcessInstances of the active ProcessTemplate of this application.

When an Application is removed, the current ProcessTemplate is removed. If the current ProcessTemplate installation has resulted in internally deprecating older versions (ProcessTemplates) of the same Application, then they are not automatically made active.
Process template management

A process is a complete unit in a workflow system. It defines the flow of the work divided into small steps called worksteps. Worksteps have an assigned performer and they can have data attached. A process with worksteps, assigned performers, and attached defined dataslots, is called a Process Template. A process template should completely describe the workflow logic and data flow. It is usually designed and made available to the system by the administrator, or a manager.

To perform a specific workflow, a normal user creates an instance from a process template, called the process instance.

For details, see the following topics:

- Accessing process templates
- Using BP Server APIs for process template management
- Creating process templates
- Retrieving process templates
- Working with process template states
- Setting the process template priority
- Managing the process template information
- Working with dynamic process templates
- Working with subprocesses
Accessing process templates

BP Server provides the following client-side Smart Value Objects (SVOs) to manage the process templates.

- ProcessTemplate
- ProcessTemplateList

Using BP Server APIs for process template management

BP Server provides object-oriented APIs for process template management. The client first needs to connect to the BP Server before executing any BP Server APIs.

Creating process templates

After a ProcessTemplate is created, it is not available until it is activated. BP Server provides a single API to create and activate a ProcessTemplate. However, you need not activate the ProcessTemplate when you create it. You can activate it later when required.

- public ProcessTemplate createProcessTemplate(Session session, String xml, boolean activate)

  At the time of creation, you can avoid activating the ProcessTemplate by setting the Boolean parameter activate to "false". Later, when required, the ProcessTemplate can be activated by invoking the activate() API.

  As an example, consider the following parameters:

  - String xml = "Hiring.xml";
  - boolean activate = true; // activates the process template at creation time
  - ProcessTemplate pt = blServer.createProcessTemplate(session, xml, activate);

  This API creates a ProcessTemplate for the Hiring application, and activate it.

  **Note:** If you try to create a ProcessTemplate which is already created, then BP Server throws an exception.

- public void activate()

  If a ProcessTemplate is created, but not activated, then you can activate it by invoking this API.

  **Note:** If you try to activate a ProcessTemplate which is already activated, then BP Server throws an exception.
An important point to note is that while an activated process template can be suspended, a suspended process template cannot be activated. A suspended process template can only be resumed. If you try to activate a suspended process template, then BP Server throws an exception.

## Retrieving process templates

BP Server provides the following APIs in the BP Server to retrieve the ProcessTemplate objects. This API returns all the `ProcessTemplate` objects of a given session.

- `public ProcessTemplateList getProcessTemplateList (Session session)`

  BP Server also provides more APIs for the `ProcessTemplate` retrieval. You can use this API to get the list of all the `ProcessTemplates`. The list obtained can be iterated to get the individual `ProcessTemplate` object.

- `public ProcessTemplate getProcessTemplate (Session session, long ptID)`

  You can get a specific `ProcessTemplate` corresponding to the process template ID provided as an input parameter.

- `public ProcessTemplate getProcessTemplate (Session session, String ptName)`

  Every `ProcessTemplate` has a name, which you can provide as an input parameter to retrieve the specific process template.

- `public ProcessTemplateList getActivatedProcessTemplateList(Session session)`

  A `ProcessTemplate` can be in different states such as activated or suspended. This API helps to get the list of all activated process templates.

- `public ProcessTemplateList getSuspendedProcessTemplateList(Session session)`

  A `ProcessTemplate` can be suspended due to some reason, or it can even be suspended internally (deprecated) because of versioning. This API returns the list of all suspended `ProcessTemplate` SVOs.

- `public ProcessTemplateList getUserProcessTemplateList(Session session)`

  This API returns a collection of the ProcessTemplate SVOs for the current session. The user information is retrieved internally from the Session object provided as an input parameter.

- `public ProcessTemplateList getUserAuthorizedProcessTemplateList(Sessionsession)`

  You can retrieve the list of all the user authorized ProcessTemplate SVOs by invoking this API. The user information is retrieved internally from the Session object provided as an input parameter.

## Working with process template states

A `ProcessTemplate` can go through different states during its life-cycle.

- Created
- Activated
- Suspended
- Removed
- Deprecated
Note: A deprecated process template is considered as *internally* suspended.

**Figure 12: ProcessTemplate States** on page 98 illustrates the states which a *ProcessTemplate* goes through.

**Figure 12: ProcessTemplate States**

---

**Getting the process template states**

BP Server provides the APIs to determine the current state of the ProcessTemplate.

- **public String getStateDescription()**
  
  This API returns the state of *ProcessTemplate*, from the possible states listed below:
  
  - P_CREATED
  - P_INSTALLED
  - P_SUSPENDED

The following direct APIs are available to check whether a *ProcessTemplate* is Activated or Suspended.

- **public boolean isActivated()**
  
  If the process template is in the Activated state, then you get the value as “true”. You can perform any operation on the Activated *ProcessTemplate*, including suspension and removal.

- **public boolean isSuspended()**
  
  If the process template is in the suspended state, then you get the value as “true”. You can only resume a suspended *ProcessTemplate*, or remove it completely.

- **public boolean isProcessTemplateExist(Session session, String ptName)**
  
  BP Server provides a direct API in BP Server, which enables you to check the existence of a process template without loading the *ProcessTemplate* SVO.
Changing the process template states

BP Server allows the transitions between suspending and resuming the process template. The process template can also be removed explicitly with the API support.

• public void suspend()

An activated process template can be suspended by invoking this API. In this state, new instances of the process template cannot be created. However, already existing process instances of the process template are not affected.

In the suspended state, attributes of the process template cannot be updated. The suspended process template can be resumed later, if required.

Note: BP Server does not allow suspending a process template that is not activated. If you try to invoke the API on a Suspended process template, then an exception is raised.

• public void resume()

You can use this API to reactivate the suspended process template. BP Server does not allow you to resume a process instance that is already activated.

• public void remove()

The API removes the Activated process template, irrespective of its state. You can remove a process template from the BP Server, if there are no process instances for that process template.

If you want to remove a process template that has one or more process instances, then you must first remove all the process instances. BP Server provides an API for removing the all process instances of that process template. Refer to the chapter Fetching the next available work item on page 55.

Setting the process template priority

The process template priority has four possible options:

• Low
• Medium
• High
• Critical

Depending on the requirement, you can also define your own priorities in the configuration file OEBPS_HOME\conf\oebps.conf. However, once the applications are installed and used, the priority options should not be changed.

If you set a priority which is not from the specified list, then at the time of using setPriority(), an exception is raised.

Note: Priority is not case-sensitive.

• public void setPriority(String Priority)

A process template has a priority associated with it, which can be set at the time of creation of process template or with the help of this API. By default, the priority is Medium.
Managing the process template information

BP Server provides APIs to manage the process template information such as attributes and dataslot information.

Setting process template attributes

You can set the attributes for a process template.

After setting the attributes of a process template, `save()` API must be invoked for effecting the changes on the server.

- `public void save()`
  
  Invoke this API after updating the attributes to reflect the modified values in server.

Retrieving process template information

BP Server supports the APIs to retrieve the information of a process template.

Getting dataslot information

A dataslot is the mechanism that allows BP Server to handle data. You may require dataslot information while working with a process template, and BP Server supports APIs to retrieve it with the help of a `ProcessTemplate` object. Refer to the chapter Dataslot management on page 139.

Working with dynamic process templates

Each process instance is a self-contained unit and can be modified dynamically at runtime without modifying the process template. For each modified process instance, a dynamic process template is created. It remains active as long as the process instance is active, and is removed after the completion of the process instance.

BP Server provides no direct BP Server API to access the dynamic process template. You can access the dynamic `ProcessTemplate` by invoking the following API on the `ProcessInstance` SVO.

- `public ProcessTemplate getProcessTemplate()`

For more information, refer to Working with dynamic process instances on page 108.

Working with subprocesses

Business Process Server supports a subprocess workstep that allows invoking a child process from the main process. This lets you develop complex processes with composition of subprocesses and also encourages subprocess reuse across multiple applications.
You can specify the template name or the application name of the subprocess as the subprocess name in the subprocess workstep of the main process template. An alias name can be provided to each subprocess. If provided, then the subprocess instance would be created with the alias name; otherwise the subprocess name would be considered.

The following rules govern the name of the subprocess:

• You can specify the process template name as the subprocess name in the subprocess workstep of the main process.

• If you specify the process template name of a deprecated process as the subprocess name in the subprocess workstep of the main process, and if there is another process template with the same application name as that of the deprecated process template which is active, then the active version is selected during the subprocess creation.

• You can specify the application name as the subprocess name in the subprocess workstep of the main process.

Similarly, the duration of a subprocess is equal to the duration of the workstep that created it. However, if the same subprocess is created independently and not by its parent process, then its duration is the duration specified in the process template.

**Subprocess alias**

Typically, a subprocess instance name consists of the parent process name, parent process ID, subprocess name, and subprocess ID. In the case of a multi-level subprocess, its name becomes very long. You can control the subprocess name length by using an alias. The format for a subprocess name with alias is as follows:

<alias>#<sub-piid>

You can define an alias for a subprocess along with the subprocess name. The alias is stored as an attribute of the *SubProcess* tag. This attribute may contain a constant value or a mapped dataslot. If the performer tag of the subprocess workstep contains hard-coded subprocess name, then the alias’ mapped dataslot must be of type CHARACTER. However, if the performer tag contains a mapped dataslot for multiple subprocesses, then the alias must be of type List. By using a LIST dataslot, each index can specify the alias for each of the multiple subprocesses. When an alias is not defined, the default value of the alias is same as the template name.

The different options for a single subprocess and multiple subprocesses are described in Table 12: Alias options for single and multiple subprocesses on page 101.

**Table 12: Alias options for single and multiple subprocesses**

<table>
<thead>
<tr>
<th>Subprocess name</th>
<th>Alias</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single Subprocess</td>
<td></td>
</tr>
<tr>
<td>Hard-coded name</td>
<td>Hard-coded alias or Mapped dataslot of type CHARACTER.</td>
</tr>
<tr>
<td>Mapped dataslot of type CHARACTER</td>
<td>Hard-coded alias or Mapped dataslot of type CHARACTER.</td>
</tr>
<tr>
<td>Hard-coded name or Mapped dataslot of type CHARACTER</td>
<td>If the alias is not provided, then the template name is used.</td>
</tr>
<tr>
<td>Multiple Subprocesses</td>
<td></td>
</tr>
<tr>
<td>Subprocess name</td>
<td>Alias</td>
</tr>
<tr>
<td>---------------------------------------</td>
<td>-----------------------------------------------------------------------</td>
</tr>
<tr>
<td>Mapped dataslot of type LIST</td>
<td>Mapped dataslot of type LIST.</td>
</tr>
<tr>
<td>Mapped dataslot of type LIST</td>
<td>If the alias is not provided, then the template name is used for all subprocesses.</td>
</tr>
<tr>
<td>Mapped dataslot of type LIST, having four entries in the list.</td>
<td>If the alias is a Mapped dataslot of type LIST with two entries, then the first two subprocesses are created with the alias based on the index. The remaining two use the template name as alias. For example, Subprocess LIST has four entries sp1, sp2, sp3 and sp4. The alias LIST has two entries als1 and als2. The template name is temp1. Then the following aliases are used for the four subprocess. Sp1 – als1 Sp2 – als2 Sp3 – Sp3 Sp4 – Sp4</td>
</tr>
</tbody>
</table>
A ProcessInstance object represents an active business process. It is a running instance of the ProcessTemplate. In addition to the created worksteps and data, it also contains a copy of the process definition, so that the instance can be modified dynamically at runtime without altering the process template.

A process instance has the following characteristics:

- Each instance is uniquely identified by a name within the system environment.
- Each instance is a self-contained unit and can be modified dynamically at runtime without modifying the process template.

For details, see the following topics:

- Using BP Server APIs for process instance management
- Creating process instances
- Retrieving process instances
- Working with process instance states
- Setting the process instance priority
- Removing process instances
- Working with dynamic process instances
Using BP Server APIs for process instance management

BP Server provides object-oriented APIs for process instance management. The client first needs to connect to BP Server before executing any BP Server APIs.

Creating process instances

You can create a ProcessInstance for a ProcessTemplate which is already activated. After the ProcessInstance is created, it is not available until it is activated. You must create and activate a ProcessTemplate before creating a ProcessInstance.

A ProcessInstance serves as the working copy of the ProcessTemplate, and can be created by:

• BLServer
• ProcessTemplate

BP Server provides a single API to create and activate the ProcessInstance. However, you need not activate the ProcessInstance when you create it. You can activate it later when required.

The state of the ProcessInstance is governed by the parameter activate (type Boolean). If you want to create and activate the ProcessInstance, you must set the parameter to true. If you want to create but not activate the ProcessInstance, you must set the parameter to false.

• public ProcessInstance createProcessInstance(Session session, String ptName, HashMap attributes, HashMap dsValues, boolean activate)

At the time of creation, you can avoid activating the ProcessInstance by setting the Boolean parameter activate to "false". Later, when required, the ProcessInstance can be activated by invoking the activate() API.

Note: The characters *, !, #, @, \, ,", /, ?, :, %, $, < and > are restricted characters that cannot be specified in the process instance name. However, underscores can be used.

Retrieving process instances

BP Server provides the class BLServer and ProcessTemplate for retrieval of ProcessInstance.

• public ProcessInstanceList getProcessInstanceList (Session session)

You can get process instances from other classes of the BP Server and from ProcessTemplate.

Retrieving process instances from BP Server

• public ProcessInstance getProcessInstance(Session session, long piid)
Every ProcessInstance has an ID that uniquely identifies each process instance of the same ProcessTemplate. You can retrieve the process instance specific to your requirement by the ID provided as an input parameter.

- public ProcessInstanceList getActivatedProcessInstanceList(Session session)
  A ProcessInstance can be in the activated state or in the suspended state. This API can be used when you require a list of all the activated process instances.

- public ProcessInstanceList getSuspendedProcessInstanceList(Session session)
  A ProcessInstance can be suspended by the user due to some reason. The API returns the list of all the suspended ProcessInstance Smart Value Objects (SVOs).

### Retrieving process instances from class ProcessTemplate

- public ProcessInstance getProcessInstance(long piid)
  You can get the process instance of the ID provided. If the process instance with the provided ID does not exist for invoking the ProcessTemplate object, then an exception is raised.

- public ProcessInstanceList getProcessInstanceList()
  Returns the complete list of all the process instances.

- public ProcessInstanceList getProcessInstanceList(long[] piids)
  Returns a list of selected ProcessInstance SVOs. This API is particularly useful when you want to reduce the overhead of retrieving all the ProcessInstance SVOs, and also do not want to retrieve ProcessInstance SVOs one by one.

### Working with process instance states

A ProcessInstance can go through different states during its life-cycle.

- Created
- Activated
- Suspended
- Completed
- Removed
Getting the process instance state

BP Server provides APIs to determine the current state of the ProcessInstance. Along with the state information, you can also check for the existence of the ProcessInstance and verify its validity.

- **public int getState()**
  Returns the state of `ProcessInstance` as an integer. BP Server defines a specific integer value for each state. This API is not generally meant to be used directly by a user. Instead, we recommend using the API `getStateDescription()`, which returns the state of `ProcessTemplate` as a String.

- **public String getStateDescription()**
  Returns the state of `ProcessInstance`, from the possible states listed below:
  - `PI_CREATED`
  - `PI_ACTIVATED`
  - `PI_SUSPENDED`
  - `PI_COMPLETED`

There are some direct APIs available to check whether a `ProcessInstance` is activated or suspended.

- **public boolean isActivated()**
  If the process instance is in the activated state, then you get the value as true. You can perform any operation on the Activated `ProcessInstance`, including suspension and removal.

- **public boolean isSuspended()**
  If the process instance is in the suspended state, then you get the value as "true". You can only resume a suspended `ProcessInstance`, or remove it completely.

- **public boolean isValidProcessInstance()**
If the process instance is valid, then you get the value as "true".

- **public boolean isProcessInstanceExist(Session session, String ptName)**

  BPServer provides a direct API in the BPServer, which enables you to check the existence of a process instance without loading the ProcessTemplate SVO.

### Changing the process instance state

BPServer allows the transitions between suspending and resuming the process instance. The process instance can also be completed explicitly with the API support.

- **public void suspend()**

  An activated process instance can be suspended by invoking this API. In the suspended state, attributes of the process instance cannot be updated. The suspended process instance can be resumed to continue further processing.

  **Note:** BPServer does not allow suspending a process instance that is not activated. If you try to invoke the API on a Suspended process instance, then an exception is raised.

- **public void resume()**

  You can use this API to reactivate the Suspended process instance. BP Server does not allow you to resume a process instance which is already activated.

- **public void complete()**

  The API completes the activated process instance. When you explicitly complete the ProcessInstance, it is also removed internally.

  **Note:** BPServer does not allow completing a process instance which is created, but not activated. It also does not support completing a suspended process instance. If you try to invoke the API on a suspended process instance, or on a process instance that is not yet activated, then an exception is raised.

### Setting the process instance priority

The process instance priority has four possible options:

- Low
- Medium
- High
- Critical

Depending on the requirement, you can also define your own priorities in the configuration file $OEBPS_HOME/conf/oebps.conf$. However, once the applications are installed and used, the priority options should not be changed.

If you set a priority which is not from the specified list, then at the time of using setPriority(), an exception is raised.
Note: Priority is not case sensitive.

- public void setPriority(String Priority)
  A process instance has a priority associated with it, which can be set at the time of creating the process instance or with the help of this API. By default, the priority is Medium.

Removing process instances

BP Server provides an API which can directly remove the process instance irrespective of its state.

- public void remove()
  You can remove a process instance that is in any state, from the BP Server.

  Note: The ProcessInstance can also be removed by invoking the API on the corresponding ProcessTemplate.

- public void removeProcessInstance(long piid)
  You must specify the ProcessInstance ID which is to be removed. If the ProcessInstance ID does not exist, then and an exception is raised.

- public long removeAllProcessInstance()
  This API removes all the process instances and returns the ID of all the removed process instances.

Working with dynamic process instances

Each process instance is a self-contained unit and can be modified dynamically at runtime without modifying the process template. For each modified process instance, a dynamic process template is created. It remains active as long as the process instance is active, and is removed after the completion of the process instance.

For example, the Hiring process template provides a generic Human Resources (HR) solution to hiring new employees. HR can use this process for considering each candidate individually. A process instance is created every time a new candidate is considered. Every workstep and attached data within an instance are related to each specific candidate. The specified process flow usually meets the general recruitment conditions. However, some special recruitments may require template modification with more worksteps or more dataslots.
Figure 14: Dynamic process instance of hiring process template on page 109 illustrates how a dynamic process instance of the Hiring process template is created.

Figure 14: Dynamic process instance of hiring process template

In this case, an additional workstep is included to assess the technical aspect of a candidate, while the general process instances also run simultaneously with the original process template definition.

However, before you attempt to modify any instance dynamically, note that:

- The selected process instance must be active and that none of the worksteps should be in the suspended state.
- All active worksteps in the process instance must be human-performed worksteps.
- The process template information is obtained through the process instance SVO.
- The process template name and the process template ID are same as the previous process template.
- The edited process instance can run concurrently with the original process instance, or the original process instance can remain suspended.

While modifying any instance, you can:

- Modify the properties of any inactive workstep by double-clicking it.
- Add a workstep to the process instance.
- Remove an Inactive workstep.
- Drag and drop existing predefined adapters or Web applications from the Task pane, or those that have been created elsewhere and added to the Task pane.
- Provide support for Swim Lanes for Performers, after the swim lanes were added in the Progress Developer Studio for OpenEdge.

Note that while you are modifying any process instance, you are not allowed to:

- Add or delete dataslots.
- Modify datslot metadata and values.
• Add or delete performers.
• Define new adapters or a subprocess.
• Add Rollback to an activity.
• Specify header and footer for a workstep.
• Modify properties of an active or completed workstep.
• Save any modifications if the process instance is suspended or has a suspended workstep.
• Use versioning with dynamic process instances.
• Complete tasks by e-mail. When the dynamic process instance is dynamically created, the performers of the activated human-performed tasks are not notified about their tasks.
• Access the dynamic process template/workstep template SVO directly.
• Create dynamic process instances of a Subprocess Instance.

**Note:** You can modify any process instance only once. If further modification is attempted, then a warning message is displayed.

BP Server provides a direct API to create a dynamic `ProcessInstance`, where the newly created dynamic `ProcessInstance` becomes the child process of the invoked `ProcessInstance`.

```java
public ProcessInstance create(java.lang.String xmlDefinition, Boolean removeOnCompletion)
```

The dynamic `ProcessInstance` and its parent process have the same `ProcessTemplate` ID with different XML definition. As a result, the user can get either the original `ProcessTemplate` on the parent `ProcessInstance` or the dynamic `ProcessTemplate` on the dynamic `ProcessInstance` SVO.
Workstep template management

Each workflow or the process template is divided into small steps called worksteps. These are the basic building blocks of the workflow processes. WorkStepTemplate is the template level object of a workstep and provides the blueprint for a WorkStepInstance.

For details, see the following topics:

- APIs for workstep templates
- Retrieving workstep templates
- Working with workstep template states
- Working with workstep template priority
- Working with workstep template attributes
- Working with workstep template dataslots
- Nested workstep with multi-subprocesses
- Accessing infopads from BP Server

APIs for workstep templates

BP Server provides appropriate APIs to determine the current type of WorkStepTemplate. Along with the type information you can check for the e-mail support provided, and verify whether the WorkStepTemplate is dynamic or not.
Chapter 17: Workstep template management

APIs

- public int getType()
  Returns the type of *WorkStepTemplate* as integer value. BP Server defines specific integer value for each type. This API is not generally meant to be used directly. Instead, it is recommended to use the API *getTypeString()*, which returns the type of *WorkStepTemplate* as String.

- public String getTypeString()
  Returns the type of *WorkStepTemplate* as a String value. The possible types of a *WorkStepTemplate*:
  - START
  - END
  - OR-JOIN
  - XOR-JOIN
  - AND-JOIN
  - PARALLEL-SPLIT
  - DECISION-SPLIT
  - EXTERNAL
  - ATOMIC
  - NESTED
  - SUBSCRIBER
  - PUBLISHER
  - INLINEBLOCK
  - BLOCKSTART
  - BLOCKEND

- public boolean hasEmailSupport()
  Checks whether the *WorkStepTemplate* is e-mail supported. If the *WorkStepTemplate* is supported by e-mail, then user can perform the task via e-mail.

- public boolean isDynamic()
  Checks whether the *WorkStepTemplate* is dynamic.

Retrieving workstep templates

*WorkStepTemplate* can be retrieved from class *ProcessTemplate* or *ProcessInstance*.

- public *WorkStepTemplate* getWorkStepTemplate(java.lang.String wsName)
  BP Server provides this API to retrieve the object of workstep template or instance corresponding to any particular workstep. The method *getWorkStepTemplate(String wsName)* in class *ProcessTemplate* or *ProcessInstance* can be used to retrieve a *WorkStepTemplate* object by providing the name of workstep. You must first retrieve the *ProcessTemplate* object, which is described in the Retrieving process templates on page 97.
• public java.util.Vector getWorkStepTemplateList()
  You can retrieve the WorkStepTemplate object for all the worksteps available with the process template or instance. This list includes InlineBlock workstep and also all activities defined within all InlineBlocks.

• public java.util.Vector getWorkStepTemplateList(boolean includeInlineActivities)
  You can retrieve the WorkStepTemplate object for all the worksteps available with the process template or instance. To include all the WorkSteps defined within all InlineBlocks, pass the parameter as “true”. If you pass it as “false”, then it include only the InlineBlock workstep defined in the main process.

Working with workstep template states

Figure 15: Workstep state transitions on page 113 shows the state transitions of a workstep.

Figure 15: Workstep state transitions

Typically, during the creation of a process instance, the workstep is in the created or inactive state. Before activating a workstep, its skip condition is evaluated. If the skip condition is "true", then the workstep execution is skipped and the next workstep gets activated. If the skip condition is "false", then the workstep is activated.

During the activation of the workstep, first the precondition is evaluated in case it exists. If the precondition evaluation returns "false", then the workstep goes to the suspended mode. If the precondition evaluation returns "true", then the pre-JavaScript is evaluated.

If the evaluation of pre-JavaScript fails, and if the compensatory action is specified, then the process instance is rolled back. If the evaluation of pre-JavaScript fails, and there is no compensatory action specified, then the workstep is suspended or stays suspended. After suspension, the resumption of the worksteps starts from the place where it was suspended. If the evaluation of pre-JavaScript succeeds, then the workstep is executed.
During the completion of the workstep, the post-JavaScript is evaluated first. If the evaluation of the post-JavaScript fails, and if the compensatory action is specified, then the process instance is rolled back. If the evaluation of post-JavaScript fails, and there is no compensatory action specified, then the workstep is suspended, or stays suspended. After suspension, the resumption of the worksteps starts from the place where it was suspended.

When a workstep is completed, its loop condition is evaluated. If the loop condition is "true", then the same workstep is activated again and the skip condition for that workstep is evaluated. If the loop condition evaluates to "false", then the next workstep is evaluated.

You can use JavaScript to embed customized business logic into Start, Activity, Subprocess, and Adapter worksteps. Using JavaScript in a workstep is helpful because:

• It executes as an Adapter workstep, without a need to write a Java program.
• It manipulates dataslots within a workstep, without a need to write an adapter.
• It performs JavaScript's regular expression, searching function and pattern matches, as well as using its math library feature to run complex calculations on dataslot values.

If a workstep has both skip and loop conditions, then skip condition is evaluated first before activating the workstep. If the skip condition is "true", then the workstep is skipped and the next workstep is activated without evaluating the loop condition. If the skip condition is "false", then the workstep is activated. When it is completed, the loop condition is evaluated. If the loop condition is "true", then BP Server evaluates the skip condition and repeats these steps. If the loop condition is "false", then BP Server executes the next workstep.

If a workstep is in a loop whose condition is always "true", then that workstep is executed in a never-ending loop. This degrades the performance of the BP Server and EJB servers. To avoid this, you can set the bpserver.ws.max.ActivationCount parameter in the bpserver.conf file to specify the maximum number of times a workstep can be executed. By default it is set to 500. After reaching this number, the workstep is suspended in the next iteration. To resume such a workstep, you must increase the bpserver.ws.max.ActivationCount parameter and reload the bpserver.conf file.

For the connector worksteps (for example, an OR join or a DECISION workstep), if the loopcounter exceeds the maximum limit, then only a warning is logged in the bpserver.log file.

The following APIs help to retrieve the information about the skip and loop conditions.

• public void getSkipCondition()
  Helps in retrieving the skip condition related to this workstep.

• public void getLoopCondition()
  Helps in retrieving the loop condition related to this workstep.

• public void getLoopConditionCounterDSName()
  Helps in retrieving the name of the dataslot being used as the loop condition counter and its value.

### Using PreJavaScript

The following APIs help to set and retrieve the pre-JavaScripts:

• public void setPreJavaScript(java.lang.String prejs)
  Sets the pre-JavaScript code related to this workstep.

• public String getPreJavaScript()
  Helps in retrieving the pre-JavaScript related to this workstep.
Using Post-JavaScript

The following APIs help to set and retrieve the post-JavaScripts:

- **public void setPostJavaScript(java.lang.String postjs)**
  Sets the post-JavaScript code related to this workstep.

- **public java.lang.String getPostJavaScript()**
  Helps in retrieving the post-JavaScript related to this workstep.

Using compensatory JavaScript code

Using the compensatory enabled process gives you an opportunity to write compensatory JavaScript code to handle a rollback request. For example, write JavaScript in a process template using rollback points for the following cases:

- Send messages or e-mail notifications before running pre- and post-functions for a second time or for restoring the process effects regarding these functions.
- Send e-mail notifications to a computer on which 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 up all the activities completed by this workstep, such as deleting generated files, or removing new database entries.

Compensatory JavaScript is executed only if a rollback is requested by a user, or by the BP Server via a workstep's execution failure.

- **public void setCompJavaScript(java.lang.String compJavaScript)**
  Sets the compensatory JavaScript code related to this workstep.

- **public String getCompJavaScript()**
  Helps in retrieving the compensatory JavaScript related to this workstep.

Using reactivate workStep

The following APIs help to set, retrieve, and remove the reactivate worksteps.

- **public void setReactivateWorkStepName(java.lang.String rwsName)**
  Sets the reactivate workstep name for this workstep.

- **public java.lang.String getReactivateWorkStepName()**
  Returns the name of the workstep, which is reactivated if the execution of this workstep fails.

- **public void removeReactivateWorkStepName()**
  Removes the reactivate workstep ID related to this workstep.

**Note:** BP Server does not support suspending, resuming, or removing workstep templates. If you try these operations on a WorkStepTemplate, then BP Server throws an exception.
Working with workstep template priority

You can specify the workstep template priority in two ways: either by reading a range of hard-coded values from the `oebps.conf` file or by using the mapped dataslot.

The workstep template priority has four default options:

- Low
- Medium
- High
- Critical

Depending on the requirement, you can also define your own priorities in the configuration file `OEBPS_HOME\conf\oebps.conf`. However, in this case, once the applications are installed and used, the priority option should not be changed.

You can map workstep priority either to an ABL CHARACTER dataslot so that you can dynamically assign a priority to a workstep.

Whenever there is a change in the workstep template priority value, you must perform a process refresh to make the changed priority effective. For example, in the case of hard-coded values, if you change the priority of a workstep from 'High' to 'Low', then you must perform process refresh. Similarly, in the case of a mapped dataslot, if you change the priority of a workstep by changing the mapped dataslot from '@stringDS'(a CHARACTER dataslot) to '@stringDSNew' (another CHARACTER dataslot), then you must perform process refresh.

**Note:** Priority is not case-sensitive.

- `public java.lang.String getPriority()`
  
  BP Server enables retrieval the priority associated with a workstep template with this API.

- `public String getMappedPriority()`
  
  BP Server enables retrieval the dynamically assigned priority associated with a workstep template with this API.

**Note:** If you change the priority of a process template, then it does not affect the priority of the corresponding Workstep Template.

Working with workstep template attributes

You can update the workstep template information by invoking the corresponding APIs on a `WorkStepTemplate SVO`.

After updating the attributes of a process template, the `save()` API must be invoked for effecting the changes on the server.

- `public void save()`
  
  Invoke this API after updating the attributes to effect the modified values on the server.
You can retrieve the workstep template information by invoking the corresponding APIs on a WorkStepTemplate SVO.

Working with workstep template dataslots

You can update the dataslot associated with a workstep template with the help of an API provided for that purpose. You can update input and output dataslots, as well as dataslot names.

You can retrieve the names of dataslot associated with a particular workstep template with the help of the API. You can get input and output dataslot names as a HashMap, and input and output dataslot list as a Vector.

Nested workstep with multi-subprocesses

Business Process Server supports subprocess worksteps that allow invoking a child process from the main process. This enables developing complex processes which include subprocesses and reuse of subprocesses across multiple applications.

Many applications, such as insurance, mortagage, and project tenders, require instantiation of subprocess instances depending on the data dynamically. For example, consider an insurance claim process. Depending on the number of witnesses, many witness subprocesses need to be instantiated by passing data selectively and composing the outputs of responses. It is very difficult to build a process that can support a dynamic number of witness subprocesses without additional workstep constructs.

Additionally, there are situations where you need to create different subprocesses depending on the runtime data. For example, when documentation for a new product is developed, a project manager selects the type of documentation needed for that product; for example, "User's Guide", "Developer's Guide", and "Programmer's Guide". Depending on the selection, that particular set of subprocesses is instantiated from a predefined list of subprocesses.

BP Server extends the nested subprocess workstep to have a mapped dataslot as a performer that can contain multiple subprocess names in a comma-separated string or a list of strings. This is done by mapping input dataslots with index mapping. This approach helps to pass selective input dataslots and obtain selective outputs to and from multiple subprocesses.

You have to perform the following two steps to use an indexed application:

• Specify a LIST type dataslot as the performer of your subprocess workstep. The values of the list should be the names of the subprocess process templates. For example, if you want to create three instances of the "ordering" process template, then the value of the LIST dataslot should be specified as [ordering, ordering, ordering]. For more information, see Working with subprocesses on page 100.

• You can pass different values to each of the subprocesses by mapping a LIST dataslot in your main process to a dataslot (such as CHARACTER or INTEGER) in your subprocesses. Each subprocess must have the same Type of dataslot as the input dataslot for this to work correctly.

For the indexed dataslots, individual elements of the LIST dataslot are provided as the input for the subprocesses. For example, AuthorIDs=[G G Marquez, A Roy]. In this case, the first subprocess has a value G G Marquez and the second subprocess has A Roy. All the remaining subprocesses receive null values.

The non-indexed dataslot is passsed as it is to all the subprocesses. For example, PublisherName, which is a non-indexed CHARACTER dataslot is sent as a CHARACTER dataslot to the Subprocesses. The publisheraddresses dataslot, which is a non-indexed LIST dataslot is sent as an LIST dataslot to the Subprocesses.
Limitations

- While specifying the performer of a subprocess workstep, you must manually enter `<LIST Dataslot Name>` in the space provided. You cannot click the "use a dataslot" because only the CHARACTER dataslots are displayed.

- You cannot use the same LIST dataslot for both Input and Output. If you attempt to use the same dataslot for both input and output, then the LIST dataslot is not updated. You must create separate LIST dataslots to send the data and receive the data.

- The data sent back to the main process from the subprocess is set into the mapped LIST dataslot in the order in which the subprocess is completed. You may introduce additional mechanisms (for example, appending the subprocess name to the return value) to keep track of the incoming values.

Accessing infopads from BP Server

You can create an infopad using BPM Events rules. An infopad is a tabular structure to store data. Each cell of an infopad can store multiple data values of different datatypes. The data in an infopad is usually updated using BPM Events rules. Infopads are useful for reporting purposes, for storing aggregate values, for storing configuration parameters, etc. Business Process Server provides access to infopad data values from various BP Server components, namely, pre- and post-Javascripts of worksteps, preconditions of a workstep, conditions on different branches of Decision worksteps.

Infopad data is stored in a database. You can use BPM Events's query service class—com.savvion.sbm.bizpulse.client.queryservice.QSInfopadService—for accessing infopad data. This class provides access to infopad data values by directly querying the database. This class can be directly referred from various BP Server components. This class provides various getter methods to which you can pass infopad name, row number, column number and slot name as parameters and the API returns the specific slot value from specific cell of the infopad.

Accessing infopads from pre and post-JavaScript

This section lists an example of infopad access from pre and post-JavaScript.

Partial code

The partial code for the rule defining the infopad is as follows:

```java
application ScriptInfopad
module ScriptInfopad_rules
initialize{
  Scriptpad := new infopad<cell{ageVal:int,nameVal:string,instCtr:int}>[1][1]("Scriptpad");
}
finalize{
  discard(Scriptpad);
}
```

Then the slot values from this infopad can be referred in a prejavascript of a workstep as shown in the following example script:

```java
function getInfopadValues() {
  var age = Packages.com.savvion.sbm.bizpulse.client.queryservice.QSInfopadService.getLongSlot("ScriptInfopad::ScriptInfopad_rules::Scriptpad", 1, 1, "ageVal");
}```
Infopad access in a precondition of a workstep

This section lists an example of infopad access in a precondition of a workstep.

Partial code

The partial code for the rule defining the infopad is as follows:

```javascript
application PreConditionInfopad
module PreConditionInfopad_rules
initialize{
    PreCondPad := new infopad<cell{numVal:int}>[1][1]("PreCondPad");
} finalize{
    discard(PreCondPad);
}

Then a precondition of a workstep can be written as:

QSInfopadService.getLongSlot("PreConditionInfopad::PreConditionInfopad_rules::PreCondPad", 1, 1, "numVal") > 10
```

Infopad access for defining a condition for a decision workstep

This section lists an example of infopad access to define a condition for a Decision workstep.

Partial code

The partial code for the rule defining the infopad is as follows:

```javascript
application DecisionInfopad
module DecisionInfopad_rules
initialize{
    DecInfopad := new infopad<cell{numVal:int}>[1][1]("DecInfopad");
} finalize{
    discard(DecInfopad);
}

Then condition for a branch of Decision workstep can be written as:

QSInfopadService.getLongSlot("DecisionInfopad::DecisionInfopad_rules::DecInfopad", 1, 1, "numVal") == 10
```
Work item management

A work item is the smallest work unit for each performer. Work items are generated from worksteps, and depend on the assigned performer and the type of assignment.

Each workstep can have a single user or a group as the performer. For a Group performer, the assignment can be to:

- All the members of a group.
- All the members of a group with a specific Role.
- Any member of a group
- Any member of a group with a specific Role.

At activation time, these worksteps create work item(s) depending on the type of its performer, as described below:

- A workstep with a single performer generates a single work item.
- A workstep with a group performer and all members assigned, creates a work item for each member.
- A workstep with a group performer and any member assigned, creates a single work item which is available to each member.
- A workstep with a group performer and any member assigned, has only one member, then the work item is directly assigned to that member.
In this case, each member gets the work item in the Available Task list. Any member of the group can assign the work item to self. When the work item is assigned, its status changes from Available to Assigned. Then the work item is in the Task List of the assigned member, and it is removed from the Available Task list.

For details, see the following topics:

- Accessing work items
- APIs for work items
- Working with work item states
- Working with work item priority
- Assigning work items
- Using dataslot information
- Completing work items
- Accessing performing apps
- Getting work items from other classes

### Accessing work items

BP Server provides the following two client-side Smart Value Objects (SVOs) to manage the work items.

- WorkItem
- WorkItemList

### APIs for work items

BP Server provides object-oriented APIs for work item management. The client first needs to connect to BP Server before executing any BP Server APIs. BP Server uses the following API to get the `WorkItemList`.

```java
public WorkItemList getWorkItemList (Session session, String user)
```

### Working with work item states

BP Server provides a number of APIs for getting the state of a work item.

- public int getState()

  Returns the state of work item as an integer value. BP Server defines a specific integer value for each state. This API is not generally meant to be used directly. Instead, it is recommended to use the API `getStateDescription()`, which returns the state of work item as String.

- public String getStateDescription()
Returns the state of work item as a String value. The possible states of a work item seen by the BP Server client are:

- I_CREATED
- I_ASSIGNED
- I_AVAILABLE
- I_SUSPENDED
- I_COMPLETED
- I_REMOVED

- public boolean isAssigned()
  This is a direct API that checks whether a work item is in the Assigned state. Returns a Boolean value that is true if the work item is assigned to the performer, false if the work item is not assigned.

- public boolean isAvailable()
  This is a direct API that checks whether a work item is in the Available state. Returns a Boolean value that is true if the work item is Available but not assigned to any performer, false if the work item is not Available.

- public boolean isWorkItemExist (Session session, long wiid)
  This is a direct API that checks whether a work item exists. Returns true if a work item exists.

## Working with work item priority

The work item priority has four possible options:

- Low
- Medium
- High
- Critical

Depending on the requirement, you can also define your own priorities in the configuration file OEBPS_HOME\conf\oebps.conf. However, once the applications are installed and used, the priority options should not be changed.

If you set a priority which is not from the specified list, then at the time of setPriority(), an exception is raised.

**Note:** Priority is not case sensitive.

- public void setPriority(String Priority)
  A work item has a priority associated with it, which can be set at the time of creation of workstep, or with the help of this API. By default, the priority is Medium.
Assigning work items

When a work item is created, it can be in the Available or Assigned state. You may need to assign an available work item, reassign an assigned work item, or make an assigned work item available. The following section describes the three possible scenarios:

Assigning an available work item

• public void assign(String performer)

  This API helps in assigning an available work item to a given performer. The performer can be an user, group of users, an external application, system or a script.

  Consider that a work item is available but not assigned. Using this API, member1 can assign the work item to self by setting the performer as member1.

  **Note:** If the work item is already assigned, then invoking this API throws an exception.

Reassigning an assigned work item

• public void reAssign(String performer)

  This API helps in reassigning an Assigned work item to a given performer. The performer can be an user, group of users, an external application, system or a script.

  Consider that a work item is assigned to performer member1 and the performer now wants to reassign the task to member2. With help of this method, member1 can reassign the Assigned work item to member2 or any other performer. After reassigning, member1 should invoke save() API to reflect the changes to server.

  **Note:** If the work item is not assigned to member1, then invoking this API throws an exception.

  It is possible to reassign the work item to the same performer. In the above example, if member1 invokes this API and sets the performer as member1, then the work item is reassigned to member1.

Reassigning bulk work items

• public void reAssign(Session session, Hashtable wiids)

  This API is supported in class WorkItem and it reassigns multiple work items to multiple performers.

• public void reAssign(Session session, List wiids, String performer)

  This API is supported in class WorkItem and it reassigns multiple work items to a single performer.

• public void reAssign(Session session, String fromPerformer, String toPerformer)

  This API is supported in class WorkItem and it reassigns all work items created for one performer to another performer.
Making an assigned work item available

• public void makeAvailable()
  This API is supported in class WorkItem and makes the work items for its workstep instance available to the remaining performers who have not yet completed it.

• public void makeAvailable(Vector performers)
  This API is supported in class WorkItem and makes the work items for its workstep instance available to the given list of performers.

  **Note:** If the performer is group all, then invoking this API throws an exception.

Using dataslot information

Dataslots help in obtaining and passing the information from one work item to another. Each workstep has input and output dataslots representing information obtained and information passed on respectively.

Getting dataslot information

A work item Smart Value Object (SVO) can obtain dataslots. The BP Server client can get a list of dataslots using the APIs provided. The DataSlotList object thus obtained can be iterated to get a single DataSlot object obtaining the complete set of dataslot supported APIs on it. The workitem APIs can:

• Get the entire list of dataslots - input as well as output.
• Get only the input or the output dataslot list.
• Get input or output dataslot name list only.

These APIs facilitate the efficient use of the server and client resources, governed by the value of an input parameter withMetaData.

When the input parameter withMetaData is set to true, the HashMap containing the values of all the attributes is preloaded from the server. Each time the user requests for an attribute, the value is retrieved from the HashMap itself. On the other hand, when the withMetaData parameter is set false, it does not contain the value for all the attributes. As the user requests to get an attribute, the corresponding value is loaded from the server and cached in the HashMap.

  **Note:** It is recommended to set withMetaData = true when you have to work on the entire set of attributes, and set withMetaData = false, when only frequently used attributes are to be processed.

• public HashMap getDataSlotList(boolean withMetaData)
  This API provides the list of dataslot (both input and output) as a HashMap, with the Key and Value, as shown in Table 13: List of dataslots as HashMap on page 126.
Table 13: List of dataslots as HashMap

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
<th>Value type</th>
</tr>
</thead>
<tbody>
<tr>
<td>INPUTDATASLOTS</td>
<td>Input DataSlotList</td>
<td>DataSlotList SVO</td>
</tr>
<tr>
<td>OUTPUTDATASLOTS</td>
<td>Output DataSlotList</td>
<td>DataSlotList SVO</td>
</tr>
</tbody>
</table>

- `public DataSlotList getInputDataSlotList(boolean withMetaData)`

  Input dataslots are the read-only dataslots. The predefined dataslots are input dataslots as their values are not editable. The input dataslots are obtained as the `DataSlotList` object.

  Similarly, `getOutputDataSlotList()` helps in retrieving the list of output dataslots.

- `public HashMap getInputDataSlotNames()`

  Each work item has some input dataslots and this API retrieves names of `InputDataSlot` in HashMap, where the key-value pair depends on the application. Key and value, both are in String format and key is case sensitive.

  Similarly, `getOutputDataSlotNames()` helps in retrieving the names of the output dataslots.

Getting metadata information

The `DataSlot` SVO has a list of attributes to manage the information. Except for the name and value, all the other attributes have been taken as metadata, as shown in Table 14: Dataslots as HashMap on page 126.

Table 14: Dataslots as HashMap

<table>
<thead>
<tr>
<th>Key</th>
<th>Value type</th>
<th>Datasslot type</th>
</tr>
</thead>
<tbody>
<tr>
<td>LABEL</td>
<td>java.lang.String</td>
<td>Any</td>
</tr>
<tr>
<td>DATASLOTTYPE</td>
<td>java.lang.String</td>
<td>Any</td>
</tr>
<tr>
<td>DATASLOTMULTILINE</td>
<td>java.lang.Boolean</td>
<td>Any</td>
</tr>
<tr>
<td>REQUIRED</td>
<td>java.lang.Boolean</td>
<td>Any</td>
</tr>
<tr>
<td>DATASLOTISGLOBAL</td>
<td>java.lang.Boolean</td>
<td>Any</td>
</tr>
<tr>
<td>DATASLOTISPUBLIC</td>
<td>java.lang.Boolean</td>
<td>Any</td>
</tr>
<tr>
<td>DATASLOTBUSINESSACCESS</td>
<td>java.lang.Boolean</td>
<td>Any</td>
</tr>
<tr>
<td>DATASLOTSIZE</td>
<td>java.lang.Integer</td>
<td>Any</td>
</tr>
<tr>
<td>DATASLOTCHOICES</td>
<td>java.util.Vector</td>
<td>Any</td>
</tr>
<tr>
<td>ISMONITORDS</td>
<td>java.lang.Boolean</td>
<td>Any</td>
</tr>
<tr>
<td>DATASLOTDEFAULTVALUE</td>
<td>java.lang.Object</td>
<td>Any</td>
</tr>
</tbody>
</table>
• public HashMap getMetaData()
  The client can invoke getMetaData() API on a DataSlot SVO to get the metadata information.

Completing work items

• public void complete(HashMap dsValues)
  You can complete an assigned work item by passing the DataSlot values. This API first assigns the DataSlot values, and then completes the work item. If the work item is successfully completed, then it is removed automatically from the Assigned list.

• public void complete()
  BP Server client can complete an assigned work item by invoking this API. If the work item is successfully completed, then it is removed automatically from the Assigned list.
  Any work item can be completed using this API, irrespective of its state. For example, if a work item is in the Suspended state, then the client can complete it using this API.

  Note: Work item removal is allowed only if the work item is not suspended.

Accessing performing apps

BP Server provides the following client-side Smart Value Object (SVO) to work with the performing applications of work item:

• PerformingApp
  The PerformingApp SVO contains the information about the performing applications (presentation types) of work item.

APIs for performing apps

BP Server provides object-oriented APIs to work with the performing applications of work item. The client first needs to connect to BP Server before executing any BP Server APIs. BP Server uses the following API to get the PerformingApp.

  public java.lang.Object getPerformingApp () throws java.rmi.RemoteException

  The getPerformingApp method returns a performing application of the WorkItem. The return value provides information about the presentation logic of the activity workstep of the WorkItem.

  A work item can have multiple presentation types, the getPerformingApp () method returns a list of performing applications. Each item of the list provides information about one presentation logic.
Working with work item performing apps

The PerformingApp SVO provides the following methods for working with the performing applications of a work item.

- `public PerformingType getType()`
  Returns an enum value defined in PerformingType enum representing the type of the PerformingApp. Enum PerformingType is defined in PerformingApp SVO and contains the following values:
  - JSP
  - WEBAPP
  - PRIVATE
  - DOTNET
  - MOBILE

- `public String getTarget()`
  Returns a String with the value of the presentation type. For example, returns the value "URL" of the Mobile presentation type.

- `public Map<String, String> getMetadata()`
  Returns a HashMap containing key-value pairs if defined as part of the presentation type metadata.

Getting work items from other classes

You can get work items from other classes of BP Server, or from classes ProcessInstance and WorkStepInstance.

Getting work items from BP Server

- `public WorkItem getWorkItem(Session session, long wiid)`
  Returns WorkItem object corresponding to the WorkItem ID. The WorkItem object thus obtained, can be used to invoke work item related APIs.

- `public WorkItem getWorkItem (Session session, String wiName)`
  Returns WorkItem object corresponding to the name of the work item. This API is deprecated and is available for backward compatibility only.

- `public WorkItemList getAssignedWorkItemList(Session session, String user)`
  The performer is assigned with various tasks, which must be completed within the estimated time. The list of work items specific to a particular user can be obtained with the help of this API.
  The list thus obtained can be iterated to get the individual WorkItem object. This object can be used to get and manipulate the work item information.

- `public WorkItemList getAssignedWorkItemListRO(Session session, String user)`
Returns Assigned WorkItemList object for a user in read-only transaction mode. The WorkItem object obtained from the list can invoke the API to get the work item information. This information is read-only and any manipulations or modification on this object cannot be done.

• public WorkItemList getAvailableWorkItemList(Session session, String user)
  The performer can have Available as well as Assigned work items in the task list. This API finds its use in retrieving all the Available (not Assigned) work items for a particular user.

• public WorkItemList getAvailableWorkItemListRO(Session session, String user)
  Returns Available (not Assigned) WorkItemList object for a particular user in read-only transaction mode. The object thus obtained can only be used to get the information. This information is read-only, and any manipulations or modification on this object cannot be done.

• public WorkItemList getWorkItemList(Session session)
  Returns both, Assigned and Available work item list object for a session.

• public WorkItemList getWorkItemListRO(Session session)
  Returns both, Assigned and Available work item list objects in read-only transaction mode for a session. The API enables you to view the complete task list of all the users in a session.

• public WorkItemList getWorkItemList(Session session, String user)
  Returns both, Assigned and Available WorkItemList object for a particular user. When the work items of a specific user are required, this API can be used.

• public WorkItemList getWorkItemListRO(Session session, String user)
  Returns both, Assigned and Available WorkItemList object for a particular user in read-only transaction mode.

Getting work items from class ProcessInstance

• public WorkItemList getWorkItemList()
  The ProcessInstance is characterized by the specific work items. The user can get the list of all work items of a process instance through this API. The individual WorkItem object is obtained on iterating the list.

Getting work items from class WorkStepInstance

• public WorkItemList getWorkItemList()
  A workstep has multiple work items when the performer level is group (ALL). The API returns a list of work item smart value objects related to this WorkStepInstance.
Workstep Instance Management

Each workflow process of the process template is divided into small steps called Worksteps. These are the basic building blocks of the workflow process. WorkStepTemplate is the template level object of a workstep and provides the blueprint for a WorkStepInstance.

The WorkStepInstance is actually executed during runtime.

For details, see the following topics:

- Accessing workstep instances
- APIs for workstep instances
- Retrieving workstep instances
- Working with workstep instance types
- Working with the workstep instance states
- Working with the workstep instance priority
- Reactivating workstep instances
- Retrieving and updating workstep instances
- Setting activation time
- Resuming workstep instances
Accessing workstep instances

BP Server provides the following two client-side Smart Value Objects (SVOs) to manage the workstep instances.

- WorkStepInstance
- WorkStepInstanceList

APIs for workstep instances

BP Server provides object-oriented APIs for workstep instance management. The client first needs to connect to BP Server before executing any BP Server APIs.

Retrieving workstep instances

You can retrieve a list of workstep instances of a ProcessInstance using the following API.

```java
public WorkStepInstanceList getWorkStepInstanceList()
```

You can retrieve a list of suspended workstep instances of a ProcessTemplate using the following API.

```java
public WorkStepInstanceList getSuspendedWorkStepInstanceList()
```

Working with workstep instance types

You can test if a workstep instance type is Activity, Adapter, or Subprocess.

- `public boolean isAtomic()`
  Returns True if the workstep instance is atomic.

- `public boolean isAdapter()`
  Returns True if the workstep instance is adapter.

- `public boolean isSubProcess()`
  Returns True if the workstep instance is subprocess.

Working with the workstep instance states

A WorkStepInstance can witness different states during its life cycle.

- Activated
- Available
- Assigned
BP Server provides the APIs to determine the current state of the WorkStepInstance.

- **public boolean isActivated()**
  
  Returns true if the WorkStepInstance is in the Activated state. You can perform any operation on the Activated WorkStepInstance, including suspension and removal.

- **public boolean isCompleted()**
  
  The workstep instance can be checked for the Completed state directly with the help of this API. If the workstep instance has been completed, then it returns true.

- **public boolean isSuspended()**
  
  If the workstep instance is in the Suspended state, then you get the Boolean value as true. You can only resume a Suspended WorkStepInstance, or remove it completely.

## Working with the workstep instance priority

The workstep instance priority has four possible options:

- Low
- Medium
- High
- Critical

Depending on the requirement, you can also define your own priorities in the configuration file OEBPS_HOME\conf\oebps.conf. However, once the applications are installed and used, the priority options should not be changed.

The WorkStepInstance priority and its WorkItems priority are the same. If you change the WorkStepInstance priority, then the priority of all the WorkItems associated with this WorkStepInstance is also changed.

If you set a priority which is not from the specified list, then at the time of setPriority(), an exception is raised.

**Note:** Priority is not case sensitive.

- **public void setPriority(String Priority)**
  
  A process instance has a priority associated with it, which can be set at the time of creation of process instance or with the help of this API. By default, the priority is Medium.

## Reactivating workstep instances

BP Server supports reactivation of a workstep instance, even after it is completed. A Completed workstep can be reactivated by setting the rollback point to true. With the help of reactivate(), the flow is restarted from the given workstep.
When the rollback point is set to true, the following steps take place to reactivate the Completed workstep.

1. Suspend the current ProcessInstance.
2. Suspend all the Activated worksteps.
3. Run compensatory JavaScript for all the worksteps from the given workstep through the last workstep in the node visited and change the path for the node visited.
4. If there are any dataslot copies, then replace all the dataslot values with default values at that point.
5. Resume the current ProcessInstance.
6. Activate the given workstep.

**Note:** Suspend action is allowed only on Activated Activity workstep, Activated nested workstep, and Message Driven workstep in Wait state. If the WorkStepInstanceList has any other combination of state/type of workstep, then a BP ServerException is raised.

## Retrieving and updating workstep instances

You can retrieve and update the workstep instance information by invoking the corresponding APIs on a WorkStepInstance smart value object (SVO). If you update the WorkStepInstance information, then this information is also updated for its associated work items.

## Setting activation time

You can set the activation time of a WorkStepInstance if it is already in activation wait state. The new value is applicable only for this workstep instance of this process instance, and not for any new process instances created.

```java
public void setActivationTime(long time)
```

The activating time is specified in milliseconds.

## Resuming workstep instances

You can resume all the suspended workstep instances of a ProcessTemplate using the following API.

```java
public resumeWorkStepInstanceList ()
```
Dynamic work item management using queues

In a business process, the performer of a human-performer workstep could be users, groups, or mapped dataslots. The list of performers may also be retrieved using an EJB method when EJB is specified as the performer of the human workstep. When a human-performer workstep is performed by ANY member of a group, the performer list is expanded to the actual list of users, and the newly created single work item is made available to the expanded list of users.

As a work item is available to a fixed number of users, the work item management is performer-centric.

However, when the members of a group are not precisely defined, performer-centric management is not useful, as the list of performers must be obtained dynamically. For example, for a field service call, you may want to assign the task to the field members available within 20 miles. In this case, the actual members available within 20 miles are to be dynamically captured from other sources of information. Further, in several cases, the performer group may be static or dynamic depending on the context, indicating that the type of group is also changeable. Consequently, BP Server must be able to manage dynamic nature of groups.

BP Server provides dynamic group modelling with work item management using queues.

For details, see the following topics:

- Push and pull mechanisms for work items
- Work item management

Push and pull mechanisms for work items

There are two basic mechanisms for work distribution or allocation in a workflow system:
Chapter 20: Dynamic work item management using queues

- **Push mechanism**: A work item is pushed to a single resource.
- **Pull mechanism**: A resource pulls work items from a common pool of work items.

BP Server supports both the mechanisms. In the Push mechanism, a single user, multiple users, or groups can be specified as performers. The Pull mechanism is supported using work item queues.

A work item queue is a dynamic data structure to which users and groups are added, allowing them to subscribe to work items assigned to the queue. Work item queues are dynamic - you can add, remove or update users and groups to a queue during runtime.

Resources can view the available work items using the work item queue. A resource could be an application, or group of users, or individual users. A work item that is marked as Available, can be seen by multiple resources at the same time. The first resource to change the state of the work item to Assigned has the work item exclusively assigned to it.

A queue may contain one or more dynamic groups, and one or more users. Work items may be available to one or more queues, and may be processed by any member of the group of users associated with these queues. With queue as performer, a work item is available only to the queue but not to any individual performer or group. Any authorized member of a queue can access all the work items available for the queue. This means performers of a work item are not static but can be dynamically associated to a queue.

Business Process Server maintains the list of queues and its dynamic groups in a database. A queue is always dynamic in nature. Each queue may contain multiple dynamic groups and users. A user may belong to one or more queues and also to many static groups. Whenever tasks for a user are requested, all the work items belonging to one or more queues of which the user is a member, are made available. BP Server authenticates the user and also verifies whether the user belongs to the queue before allowing the user to complete the task.

**Work item pull**

There are two ways to pull the work items from queues. The first approach is: when a user User1 logs in from the specified user-realm (such as LDAP, NIS, or JDBC), find all the groups that User1 belongs to, then find all the work items available for the queues with User1 and all the groups to which User1 is a member.

Sometimes external applications need to get all the work items available from one or more queues, and display the list to the members of a queue. In this approach, the external application authenticates the users to access the available work items from one or more queues. The external application may even get the work items list for a queue rather than for a user.

The API that can be used to get a work item list for a queue is

- `List getQueueWorkItemList(Session, String[], String [])`

This option is available for any value of `oebps.queues`.

**Work item management**

For any work item queue, one or more performers can be specified as a comma-separated list. You can also specify performers as a mapped dataslot or as an EJB method.

**Note**: Queue names can be specified only for human-performer worksteps with group-by-any option.
In a group-by-any workstep, the performer list may contain a list of users, groups, and queues. At the time of activation, all the queues specified in the performer list are not expanded. Instead, the work item is available for the queue itself by creating a record in the \textit{BP Server \_AvailableWorkItem} table with performer as the queue name.

\textbf{Performer List:} user1, group1, queue1, queue2, queue3

If group1 contains five users, then for each queue, one record is created in the \textit{BP Server \_AvailableWorkItem} table with \textit{AvailableFor} as the queue-name. For the above performer list, there are a total of nine records: user1, five users from group1, queue1, queue2, and queue3.

\textbf{Note:} When an available work item is assigned to one user, the records are not removed from the \textit{BP Server \_AvailableWorkItem} table, so that the work item can be made available again to the original group.

\section*{Retrieving available work items}

Whenever a user logs in, the list of all groups to which the user belongs is retrieved from the specified user management system. The list of all the queues, of which the retrieved groups were members, is also fetched using the QueueResource table. Some of these groups may not be members of any queues. The final list includes the users, list of queues (of which some groups are the members), and also the list of groups that do not belong to any queue. This list is used to search the available work items.

For example, consider a case where user1 belongs to group1, group23, group29, and group61. Also, group1 is a member of queue1, group23 is member of queue16, and group61 is member of queue15. Note that group29 does not belong to any queue. Now the final list is user1, queue1, queue16, queue15, and group29. BP Server dynamically expands this list to get the list of real performers.

\textbf{Note:} If a user belongs to more than one queue, then use custom filtering for the work item ID to avoid the work item duplication.

\section*{Specifying timer action for available work items}

When the assigned work item of a group-by-any workstep is not completed before the specified due date, the application may need to make this work item available for other performers. While designing a process template, the timer action MAKEAVAILABLE can be specified as the overdue action after the due date. When the work item is not completed before the due date, BP Server makes it available to the specified performers.

\section*{Completing assigned work items}

The dynamic nature of a queue can be well utilized until the work item is in the Available state. Once an available work item is assigned to a specific user, then any change to a queue is not effective immediately for the work items in the Assigned state.

Whenever a user completes a work item, all the groups to which the user belongs are validated against the performer list of its workstep. If the user’s current list of groups does not belong to any of the queues mentioned in the workstep performer list, then BP Server aborts the work item completion task and notifies the user with proper exception messages.

\textbf{Note:} Validation of a user association with a queue at the time of work item completion may affect the performance.
Removing user / group from a queue

When you remove a user from a queue, it immediately impacts all the new work items and work items in the Available state. However, it affects the work items in the Assigned state only when the assignee is completing the task. When a user tries to complete an assigned task that is no longer available because the user is removed from the queue, BP Server cancels that operation and notifies that the user cannot complete that task.

Removing a queue

When you remove a queue, you must move any existing available work items of that queue to another queue. A queue cannot be deleted if there are any active available work items for that queue.

When you remove a queue, the creation of new work items may fail if the deleted queue names were not replaced properly in the process templates. Before deleting a queue, it is required that process templates are modified to replace the queue to be deleted with the new queue name and also move the active available work items of a queue to another queue. However, if the objective is to remove a queue, then it can be done only when there is no reference to this queue in any of the available work items.

Note: BP Server should be active when you remove a queue.
Dataslot management

A dataslot is the mechanism that allows BP Server to handle data. It is a container for data, attached to both a workstep and a process. Since data in a process might be shared, a dataslot is like a pipe tapping into a reservoir of data.

Each process defines a set of dataslots for passing data between worksteps. Each dataslot has a Name, a Data Type, a Value and an optional list of possible values. The dataslots can be shared among many worksteps in the same process.

A workstep may not have any input dataslots attached to it, or it may have a number of input dataslots attached. Similarly, it may not have any output dataslots attached to it, or it may have a number of output dataslots attached.
At runtime, the BP Server Engine passes data to a workstep and gets the result (if any) after completion. In addition to defining the type of dataslot, you can also define a set of possible values for a dataslot, such as: Approved, Not Approved, and Request not Complete. The set of possible values helps in controlling the user input.

BP Server supports the following type of ABL datatypes as dataslots:

- CHARACTER
- INTEGER
- INT64
- LOGICAL
- DATETIMETZ
- DECIMAL
- HANDLE
- ROWID
- RAW
- MEMPTR
- DATASET
- TABLE
- OBJECT
- LIST
- MAP
BP Server provides the APIs in DataSlot and DataSlotTemplate Smart Value Objects (SVOs) to check the type of a dataslot.

Note that the following are the reserved keywords and no dataslot should be created with any of these reserved keywords:

CREATOR, PROCESSNAME, WORKSTEPNAME, PRIORITY, STARTTIME, ALLDATASLOTS, PROCESSTEMPLATENAME, PROCESSTEMPLATEID, PROCESSINSTANCENAME, PROCESSINSTANCEID, WORKSTEPNAME, WORKSTEPID, TYPE, LOOPCOUNTER, PERFORMER, PERFORMERVALUE, STARTTIME, ENDTIME, DUEDATE, DURATION, INSTRUCTION, PRIORITY, GROUPTYPE, APP_NAME, INLINEBLOCK_NAME, LOOPCONDITIONCOUNTERDS, GLOBAL_SLOT_NAME_LIST, RPID, WORKITEMNAME, WORKITEMID, PREV_PERFORMER, PREVIOUSSTATUS, RESPONSE, PI_DEBUG, PIID

For details, see the following topics:

• Using BP Server APIs for dataslot management
• Using dataslots
• Using DataSlotTemplate
• Working with different types of dataslots

Using BP Server APIs for dataslot management

BP Server provides object-oriented APIs for dataslot management. The client first needs to connect to BP Server before executing any BP Server APIs.

Using dataslots

A process instance has a number of worksteps and the worksteps are attached to the dataslots. The following sections describe how you can retrieve the dataslots attached to a process instance, and get dataslot information.

Retrieving dataslots

You can retrieve dataslots from classes BLServer, ProcessInstance, and WorkItem.

BP Server provides the API for retrieving the object DataSlot corresponding to a particular dataslot.

• public DataSlot getDataSlot(java.lang.String dsName)

This method in class ProcessInstance retrieves DataSlot object by providing the name of dataslot. You must first retrieve the ProcessInstance object.

**Note:** If the process template does not exist, then it is first created and activated. Then the process instance is created and the ProcessInstance object is retrieved.

This method returns "Not Public" for private dataslots.

• public DataSlotList getDataSlotList()
This method is useful when you want to retrieve the *DataSlot* object for all the dataslots available with the process instance.

**Getting dataslot information**

The *AbstractDataSlot* class provides APIs for getting information of the *DataSlot* or the *DataSlotTemplate*. The *DataSlot* object contains a list of attributes to manage the information. Except for the name and value, all the other attributes have been taken as metadata. You can retrieve the metadata from the *DataSlot* SVO.

**Using DataSlotTemplate**

The relationship between *DataSlot* and *DataSlotTemplate* is similar to the relationship between *ProcessInstance* and *ProcessTemplate*. The *DataSlot* is the instance level object holding the run-time value of *ProcessInstance* data, while the *DataSlotTemplate* is template level object, which stores the metadata of data. The metadata includes the attributes of data such as name, label, type, size, isPublic, isGlobal, and so on.

In addition to providing the APIs for getting run-time value of data, *DataSlot* object also provides the API to retrieve the metadata. Since metadata-related APIs are common to both *DataSlot* and *DataSlotTemplate*, they are moved in common super class *AbstractDataSlot*. However, the *AbstractDataSlot* is completely transparent to the user.

Once the *DataSlot* instance is created and associated with a *ProcessInstance*, its metadata cannot be changed. Therefore the *DataSlot* object does not provide the API to update the metadata of the *DataSlot*. On the other hand, the *DataSlotTemplate* provides the APIs to update some of the metadata.

**Retrieving DataSlotTemplates**

- public Vector getDataSlotTemplateList()

  BP Server helps in retrieval of *DataSlotTemplate* smart value object from class BLServer and *ProcessTemplate* using this API.

**Setting attributes of DataSlotTemplate**

You can set the attributes of metadata for *DataSlotTemplate*.

**Saving attributes of DataSlotTemplate**

- public void save()

  *DataSlot*, *DataSlotList* and *DataSlotTemplate* SVOs provide this API, which is to be invoked after updating the metadata to reflect the modified values in server.
Working with different types of dataslots

BP Server supports a number of different types of dataslots. This section describes the methods for some of them.

Document dataslots

A document type of dataslot is a dataslot to which the users can attach one or more documents. The documents can be uploaded to the server and the other performers can download, read and update them. You can attach any type of file to a Document dataslot. The dataslot must be specified as an output dataslot in a workstep to provide the ability to attach a file.

The default value for a Document dataslot can be specified using:

- the "file://" protocol
- the "http://" protocol
- documents at OEBPS_HOME\ebmsapps\<ptName>\docs.

When documents specified for default values are referred to, they can be a simple file name (readme.doc) or a file name with relative path (myfolder/readme.doc).

When the specified documents are not found, or are not readable, or have problems while uploading, the errors are logged but ignored so that the template or instance create operations are successful.

Note: In the case of file and http protocols, when the data is migrated to another computer, the default values should be modified to reflect the new host-name and port number.

You can use Developer Studio to configure Document dataslot to attach a single document or a bundle of multiple documents. But Business Process Portal does not allow you to attach multiple documents to a single Document dataslot. A Business Process Portal user has to remove the existing document before attaching a new document to a single Document dataslot. However, BP Server does not differentiate between single and multiple Document dataslots. Therefore, you can add more than one documents to a single Document dataslot using BP Server API. To achieve a single document behavior while using BP Server APIs, you must invoke remove method to remove the existing document before attaching a new document to a single Document dataslot. The same behavior is seen from BP Server Admin also.

The length of path of any document or folder in Document Management System (DMS) is limited to 255 characters. Therefore, the maximum permissible length for the name of any document or a folder is less than 255 characters. The maximum length of any document or a folder name is 255 minus the length of path of the parent folder.

You can impose restrictions on the type of files that users can attach to the document dataslot by using the bpmportal.document.exclude.extensions parameter from the OEBPS_HOME\conf\bpmportal.conf file. By default it is set to .exe, .cmd, .js, and .bat. You can use this feature to enhance security.

For more information on Document dataslots, refer to Application Developer’s Guide.

Adding documents

Use this method to create a new document for the specified performer with the specified document name. The content of the file is stored as the document content.


Use this method to create a new document for the specified user with the specified document name. The content of the InputStream is stored as the document content.

- public com.savvion.sbm.dms.svo.Document create(java.lang.String user, java.lang.String docName, java.net.URL srcURL)

Use this method to create a new document for the specified user with the specified document name. The content of the URL is stored as the document content.

Retrieving documents

- public com.savvion.sbm.dms.svo.Document getDocumentById(java.lang.String docId)
  You can get the Document with the specified document id.

- public com.savvion.sbm.dms.svo.Document getDocumentByName(java.lang.String docName)
  You can get the Document with the specified name.

- public java.util.List<java.lang.String> getDocumentNames()
  You can get the list of document names in the document dataslot.

  You can get the List with the name specified in the docNames array.

- public java.util.List<com.savvion.sbm.dms.svo.Document> getList()
  You can get the List of documents for the document dataslot.

  You can get the list of documents belonging to the specified creator.

Saving locally

- public void saveToFile(java.lang.String documentName, java.lang.String filePath)
  You can store the content of the specified document name at the specified file path.

- public void saveToLocation(java.lang.String documentName, java.lang.String folderPath)
  You can store the content of the specified document name at the specified folder path.

Removing documents

- public void removeById(java.lang.String docId)
  You can remove the document with the specified document id.
• public void removeByName(java.lang.String docName)
  You can remove the document with the specified document name.

• public void removeDocuments()
  You can remove all the documents of the document dataslot.

Miscellaneous
• public byte[] getContent(java.lang.String documentName)
  You can get the content of the specified document as byte array.

• public int getCount()
  You can get the number of documents contained in the Document dataslot object.

• public boolean isEmpty()
  This API returns true if the document dataslot doesn't contain any document.

• public boolean isNull()
  This API returns true if the document dataslot folder is not created.

• public com.savvion.sbm.dms.svo.Document replaceContent(java.lang.String docName, byte[] content)
  You can replace the content of the document with the specified content.

Storing document dataslot attachments in external location
Typically, the documents attached to Document dataslot are stored in a Business Process Server (BP Server) proprietary folder structure. However, you can change this location and store documents attached to Document dataslot to an external location as per your requirement in any DMS.

You can change the Document dataslot attachment’s location to a custom folder structure in a DMS in following scenarios:
• In any BP Server and Web (BPM Web Flow) application
• In a Start workstep before creating an instance
• In any intermediate activity or adapter workstep
• In a standalone Web (BPM Web Flow) application

To allow document storage of a document dataslot to a custom path:
1. You must set the oebps.dataslotstorage.custompath.enable parameter in the OEBPS_HOME/conf/oebps.conf file to true (by default, it is set to false).

   When this parameter is set to true, the application developer can set the document path for the Document dataslots using the pre or post scripts. BP Server provides functions/methods to achieve this.

2. You must pass the Document dataslot name and document path expression to these functions.
   The document path expression may contain the Character and/or Integer dataslots.

   The values of these dataslots are evaluated at runtime and replaced in the path expression to get the actual path. The script function takes care of creating the folder structure (if it is not already created) and setting its location as the path for the Document dataslot passed.
On subsequent document attachments to that dataslot, the documents are stored in the folder location specified in the script function. Note that if a document is attached to a dataslot before executing a script, then that document is stored in the BP Server proprietary folder structure.

**Note:** After you set a folder path for a Document dataslot, you cannot change it again.

**Note:** The Archiver cannot purge the documents saved in a custom location. Also, after you uninstall a business process template, the documents having custom path are not purged.

**Limitations**
The custom document path is not supported for the Document dataslots containing default documents. If the same custom folder path is used for a Document dataslot in more than one process instances, then the the documents attached to such dataslots are stored in the same folder. Therefore, when multiple users simultaneously edit the same document in subsequent worksteps, the document is overwritten, and only the changes made by the last user are stored.

**Examples**
This section covers examples of how to use script to set custom path for Document dataslot:

- For BP Server application, to attach documents to intermediate worksteps, run the following script in the Start post-script or in pre/post-script of the Activity workstep that is occurring before the workstep that has docDS as its input/output field.

  ```javascript
  jst.setDocumentPath("docDS",
  "companySales/products/${productName}/${orderId}", false);
  ```

  where **productName** is a String dataslot and **orderId** is a Number dataslot.

- For Web (BPM Web Flow) application, to attach documents to intermediate worksteps, run the following script in pre/post-script of Activity workstep that is occurring before the workstep that has docDS as its input-output field.

  ```javascript
  UtilityBean.setDocumentPath("docDS",
  "companySales/products/${productName}/${ordered}", bean, false);
  ```

  where **productName** is String dataslot and **orderId** is a Number dataslot.

- To attach documents to the Start workstep, Start workstep must use a private or public Web (BPM Web Flow) application. In the Web application, you can run the script in post-script in ‘Activity 1’ and in the next activity, you can attach the documents to the dataslot.

**Supporting multiple DMS user credentials**
By default, BP Server uses a single DMS user account for creating, updating and deleting the documents in the Document dataslot. However, Business Process Server also supports following scenarios:

- Every Business Process Server user has his/her own DMS account (one Business Process Server user to one DMS user).
- A group of Business Process Server users use a single DMS account (many Business Process Server users to one DMS user).

**To enable multi-DMS user account support**
1. You must set the following parameter in the `OEBPS_HOME\conf\SDMS.properties` as shown below.

   ```text
   dms.multiuser.credential=true
   ```

   When you set this parameter to true, Business Process Server uses DMS user name and password associated with the Business Process Server user to connect to the DMS, instead of using DMS user name and password specified in the `dms.properties` file. Every Business Process Server user can have his/her own DMS user name and password, which can be different than his/her Business Process Server credentials. This allows many Business Process Server users to share one DMS account. This provides the “many Business Process Server users to one DMS user” capability.

2. To enable one Business Process Server to one DMS user mapping, you must set the following parameter in the `OEBPS_HOME\conf\dms.properties` file as shown below.

   ```text
   dms.appuser.credential=true
   ```

   When you set this parameter to true, Business Process Server user’s user name and password is used to connect to DMS. This parameter is effective only when the `dms.multiuser.credential` parameter is set to true. This provides the “one Business Process Server user to one DMS user” capability.

   **Note:** DMS does not authenticate user name and password while creating a DMS session because it does not have its own user management. Therefore, Business Process Server users can connect to DMS with any user name.

   The following table shows the behavior of DMS with different combinations of values of the above parameters.

<table>
<thead>
<tr>
<th><code>dms.multiuser.credential</code></th>
<th><code>dms.appuser.credential</code></th>
<th>Connects to DMS using</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>false</td>
<td>User credentials specified in the <code>dms.properties</code> file.</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>DMS user credentials associated with BM user</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>Business Process Server user</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>User credentials specified in the <code>dms.properties</code> file.</td>
</tr>
</tbody>
</table>

### INTEGER dataslots

BP Server provides the following APIs for the INTEGER type of dataslot.

- **public java.lang.String getType()**
  
  You can use this API to get the String value of type of this INTEGER dataslot.

- **public java.math.BigDecimal getValue()**
  
  You can use this API to get the BigDecimal representation of this integer value.

- **public java.lang.String getStringValue()**
  
  You can use this API to get the string representation of this integer value.
• public int getPrecision()
  You can use this API to get the precision in integer format.

• public int getScale()
  You can use this API to get the scale in integer format.

• public void add (java.lang.String sdv)
  You can use this API to add string value to this integer value.

• public void add (long dv, int scale)
  You can use this API to add value to this number, where value is given in long and scale in integer format.

• public void multiply (java.lang.String sdv)
  You can use this API to multiply the value in string format to this INTEGER dataslot.

• public int compareTo (java.lang.String sdv)
  You can use this API to compare the string value with this INTEGER dataslot.

DATETIME-TZ dataslots

DATETIME-TZ dataslot is defined as a dataslot representing a specific instant in time, measured down to seconds. It converts between string and date-time format. For getting the DateTimeTZ SVO, you must create the instance of DateTime object. BP Server does not provide any API that directly returns the DateTime type object. The DateTimeTZ SVO helps the operations on DATETIME-TZ type of dataslot.

BP Server provides the following API supported by the DATETIME-TZ type of dataslot.

DateFormat is an abstract class for date/time formatting subclasses which formats and parses dates or time in a language-independent manner. The date/time formatting subclass, such as SimpleDateFormat, allows for formatting date to text, parsing text to date, and normalization. The date is represented as a Date object or as milliseconds since January 1, 1970, 00:00:00 GMT.

DateFormat provides several class methods for obtaining the default date/time formatters based on the default or a given locale and a number of formatting styles. The formatting styles include FULL, LONG, MEDIUM, and SHORT. DateFormat helps to format and parse dates for any locale. You can keep your code completely independent of the locale conventions for months, days of the week, or even the calendar format (lunar / solar).

• public java.lang.String getType()
  Using this method, you can get the type in string format.

• public java.sql.TimeStamp getValue()
  You can get the TimeStamp value of this class.

• public java.lang.String getStringValue()
  Use this API to get the String value of this class.

• public java.lang.String getStringValue(int dateStyle, int timeStyle)
  You can get the String value of this class with the given date and time formatting style. The formatting style can be defined as:

  • DateFormat.SHORT which is numeric, such as 12.13.52 or 3:30pm.
- DateFormat.MEDIUM which is longer, such as Jan 12, 1952.
- DateFormat.LONG which is longer, such as January 12, 1952 or 3:30:32pm.
- DateFormat.FULL which is completely specified, such as Tuesday, April 12, 1952 AD or 3:30:42pm PST.

- public long getTime()
  You can get the date in long format.

OBJECT dataslots

An OBJECT type dataslot is defined as a dataslot containing a serializable Java Object. OBJECT dataslot types include Vector and Array, which are passed as input or output dataslots to an Adapter. OBJECT dataslots are provided so that any Java object of any class can be associated with a process instance. The class for the Java object can be from JDK, for example, java.util.ArrayList, or it can be any custom coded java class.

Note:
If an application contains OBJECT dataslot, then before creating instances of that application, copy the class(es) for the object dataslot in the following locations.

- $webapp_dir/WEB-INF/classes
- $oebps_home/ebmsapps/common/bo/classes

Dynamic class loading for object dataslot

Business Process Server provides a functionality to allow modifications to an Object dataslot’s class and deploy such a modified class when the EJB application server is up and running. In other words, in current version of Business Process Server, there is no need to restart EJB application server to use a newer version of Object dataslot’s class.

Current infrastructure of dynamic class loading for adapter classes is used to provide dynamic class loading for Object dataslots. Thus, the folder locations to place class files for Object dataslots are same as those used for adapter classes. For more information, see Dynamic class loader for adapters on page 61. When a process template gets installed, the class of the Object dataslot is loaded for the first time. If a class is modified at runtime, then it is loaded during the next process instance creation.

Since Object dataslot’s object instance is serialized and stored in a database, it is strongly recommended to define a private static member variable as shown below in the Java class of Object dataslot:

"private static final long serialVersionUID = 7526472295622776147L; "

The long value can be obtained by executing serialver.exe from JDK\bin folder providing fully classified class name, for example, serialver com.savvion.MyObject.

Whenever class definition is changed, one should make sure that the long value for this field remains the same.

If the Object dataslot’s class is dependent on one or more other classes, then such classes can be placed in extracted format in ebmsapps folder or they can be placed in the system classpath.

Managed adapters’ classes are loaded from the system classpath. Hence, all dependent classes are also loaded by system classloader. If managed adapter invocation is the first operation after server startup, then dynamic class loading does not happen for the Object dataslot classes that are input/output to the managed adapter, since it is already loaded by system classloader.
If an Object dataslot is used in the alert condition code, then the Object code must be placed in the
%OEBPS_HOME%\ebmsapps\common\bo\classes folder.

Limitations

• If the class file of an Object dataslot is part of a JAR file, then reading of the Object dataslot using APIs
  causes a ClassNotFound or De-Serialization Exception. Therefore, it is recommended that Business Process
  Server applications should not use JAR files for Object dataslots.

• For dynamic loading of Object dataslots, keep Class files under the <OEBPS_HOME>\ebmsapps folder.

• To achieve dynamic class loading of Object dataslot’s classes, set the bpserver.classloader.dynamic
  flag in bpserver.conf to True.

• If new Object dataslots are added during process replacement, then you are required to restart the EJB
  application server. This is because already cached prepared statements becomes invalid for any schema
  changes.

LIST dataslots

The LIST dataslot type is a feature that enables you to create a dataslot that contains a list of options. In other
words, the LIST type of dataslot treats a collection of related data items as a single entity. For example, a LIST
dataslot can be used to hold options for a CHARACTER dataslot where the options are dynamically determined
by querying a database.

The main advantages of a LIST dataslot are:

• At runtime, the elements of a LIST dataslot can be accessed directly through basic actions to the provided
  adapters.

• If the LIST dataslot is mapped to a database table, then the values are read directly from database and no
  additional action is needed. In the case of the CHARACTER dataslots mapped to a database, concatenation
  of all the values coming from the table into one long string with separator characters is needed.

CHARACTER dataslots

Typically, when the CHARACTER dataslot value exceeds the allowed limit, an exception is raised. In some
special cases, if you want to truncate the value to the allowed limit, then you can set the parameter
bpserver.dataslot.value.truncate in the file bpserver.conf, to true. The CHARACTER dataslot
value is truncated and a warning message is added to the file bpserver.log.

MAP dataslots

MAP Dataslot type is a feature that can be used to store data which is in the form of key-value pair.

MAP Dataslot can store any object which implements java.util.Map and java.lang.Serializable
interfaces. You can use implementation of java.util.Map available with standard distribution of java like,
java.util.HashMap, java.util.LinkedHashMap, java.util.Hashtable, etcetera. You can also
use custom implementation of java.util.Map interface.

By default, BP Server uses java.util.LinkedHashMap for MAP dataslot.
Configuring performers

This chapter describes how you can configure the performers in Business Process Server.

For details, see the following topics:

• **Remote JNDI external performer application**
• **Local JNDI external performer application**
• **Callback code example**
• **EJB performer in atomic workstep**

Remote JNDI external performer application

Read on to configure BP Server to communicate with an external performer that is an Enterprise Java Bean deployed on a remote computer.

Consider two machines "Diablo" (local computer) and "Copper" (remote computer) with a JBoss installation. Business Process Server is installed on "Diablo." The external performer (*RemoteJNDIEP.ear*) is deployed on "Copper."

Configuring remote JNDI properties in oebpsjndi.properties file

To invoke a remote subprocess assigned as a performer for a workstep, you must configure the remote server properties in the `oebpsjndi.properties` file on the Business Process Server (BP Server).

To configure the remote server properties in the `oebpsjndi.properties` file:
1. Open the OEBPS_HOME\oebpm\server\conf\oebpsjndi.properties file in a text editor.

2. Copy the following group of properties of your remote server that you want to add in the oebpsjndi.properties file on the BP Server:

```
oebps.remotename.factory.initial=org.jnp.interfaces.NamingContextFactory
oebps.remotename.provider.url=jnp://<RemoteMachineName>:16003
oebps.remotename.principal=guest
oebps.remotename.credentials=pF8V/2DrWnw=
oebps.remotename.jndi.ejb.prefix=ejb/
```

**Note:** Where `<RemoteMachineName>` is the machine name of your remote server.


4. Save the oebpsjndi.properties file on the BP Server.

5. Restart the BP Server.
A sample `oebpsjndi.properties` file updated with remote server properties is shown below:

```properties
# Properties for JBoss app. server
# =====================================
# Properties for JBoss app. server
oebps.jboss.factory.initial=org.jnp.interfaces.NamingContextFactory
oebps.jboss.provider.url=jnp://nbhydshsingh-7:16003
oebps.jboss.principal=admin
oebps.jboss.credentials=aWbTNRj73MM=
# Appserver specific version should be removed from SBM 9.1,
# Since new common property (oebps.application.server.version) is added.
oebps.jboss.version=5.1.0
oebps.jboss.ejb.server.name=ejbServer
oebps.jboss.portal.server.name=portalServer
oebps.jboss.jndi.server.local.prefix=java:comp/env/
oebps.jboss.jndi.server.local.ejb.prefix=ejb/
oebps.jboss.jndi.ejb.prefix=ejb/
# Properties for JBoss jms server
oebps.jboss.jms.factory.initial=org.jnp.interfaces.NamingContextFactory
oebps.jboss.jms.provider.url=jnp://nbhydshsingh-7:16003
oebps.jboss.jms.principal=admin
oebps.jboss.jms.credentials=aWbTNRj73MM=
oebps.jboss.jms.remote=true
oebps.jboss.jms.factory.queue.jndiname=QueueConnectionFactory
# The below specified remote server properties will be used for invoking a remote subprocess
oebps.remotename.factory.initial=org.jnp.interfaces.NamingContextFactory
oebps.remotename.provider.url=jnp://nbhydshsingh-7.apac.progress.com:16003
oebps.remotename.principal=guest
oebps.remotename.credentials=pF8V/2DrWnw=
oebps.remotename.jndi.ejb.prefix=ejb/
```
Adding a remote external performer workstep to the process template

In the Performer box of the Properties dialog box for the workstep, the URL of the external performer is specified in the following format:

```
jndi://$RemoteMachine/<jndiName>
```

For example:

```
jndi://$copper.ejb/RemoteJNDIEP
```

Creating and deploying the ear for the remote external performer

The external performer is an EJB (RemoteJNDIEP.jar) and contains the home, remote and bean classes. The deployable ear RemoteJNDIEP.ear is created by adding RemoteJNDIEP.jar and bpsclient.jar. Add bpsclient.jar to the ear in order for the external performer to call back the BP Server.

**Note:** The RemoteJNDIEP.jar should contain a manifest file with bpsclient.jar in the classpath entry. This deployable ear file is deployed on "Copper" (remote computer).

Local JNDI external performer application

This section explains how to configure and execute a local external performer (LocalJNDIEP.ear).

Adding a local adapter workstep to the process template

In the Performer box of the Properties dialog box for the workstep, the URL of the external performer is specified in the following format:

```
jndi://$DEFAULT/<jndiName>
```

Where $DEFAULT points to the configuration details of the default application server in oebpsjndi.properties and jndiName is the JNDI name of the external performer. For example:

```
jndi://$DEFAULT/ejb/RemoteJNDIEP
```

After completing this step, deploy the LocalJNDIEP.ear.

Mapping the JNDI name of the external performer to a fully qualified class name

The <jndiName> is mapped to the actual class name in oebpsjndi.properties as shown below.

```
map.<jndiName>=<fully-qualified-class-name>
```
Callback code example

The following example displays a snippet of code to completeCallerWS for asynchronous local/remote adapters. The external performer retrieves the properties to call back BP Server from the PAKcallerID method.

```java
BP ServerManagerHome home = (BP ServerManagerHome)SBMHomeFactory.lookupHome(
    new HashMap(props),
    Class.forName("com.savvion.ejb.bizlogic.manager.BP ServerManagerHome"));
BP ServerManager manager = (BP ServerManager)home.create();
String ses = manager.connect("admin","admin");
manager.completeCallerWS(ses, pi_name, ws_name, dataslot_values);
manager.disConnect(ses);
```

EJB performer in atomic workstep

BP Server supports the EJB Performer in atomic worksteps (Activity worksteps), where the performer of the workstep is decided at runtime by executing a method on an EJB.

Performer string

The EJB performer of an Activity workstep provides both the EJB name and the method name. It has the following syntax:

- `jndi://<server-id>/<jndi name of EJB>:<method name>`
- `jndi://$<alias-name>/<jndi name of EJB>:<method name>`

Where:

- `jndi://` is the name of protocol, which is case sensitive,
- `<server-id>` or `$<alias-name>` is the name of the application server defined in `oebpsjndi.properties`,
- `<jndi name of EJB>` is the jndiname of the EJB object used to locate the EJB in the application server, and
- `<method name>` is the name of the method in the EJB remote interface, that is called by the BP Server to get the performers for the Activity workstep.

Application server

The BP Server requires the initial context factory name, URL, principal and credential of the application server to communicate to the EJB. You can provide these properties in `oebpsjndi.properties` file using any name for the application server. Use this name in the performer string as `<server-id>`. 
Example of <server-id>:

For the performer of format jndi://<server-id>/<jndi name of EJB>:<method name>, you need to specify the following entries in oebpsjndi.properties file, located in OEBPS_HOME\conf folder.

- oebps.<server-id>.factory.initial=<fully qualified class name of initial context factory>
- oebps.<server-id>.provider.url=<jndi url>
- oebps.<server-id>.principal=<username>
- oebps.<server-id>.credentials=<password>

You can also provide an alias-name in the oebpsjndi.properties file for the application server name used to describe the properties, and then use this alias name in the performer string as <alias-name>. The alias-name hides the details of the context factory, URL, principal and credential from the performer string and thus isolates the process definition from communication channel.

Example of <alias-name>

For the performer of format jndi://$<alias-name>/<jndi name of EJB>:<method name>, you need to specify the following entries in oebpsjndi.properties file:

- oebps.myappserver.factory.initial=<fully qualified class name of initial context factory>
- oebps.myappserver.provider.url=<jndi url>
- oebps.myappserver.principal=<username>
- oebps.myappserver.credentials=<password>
- <alias-name>=myappserver

BP Server also provides an additional facility to connect to the EJBs that are deployed on the same application server instance on which the BP Server is running. For this purpose, the server-name/alias-name is predefined as DEFAULT and the format of performer string is as follows:

jndi://$DEFAULT/<jndi-name of EJB>:<method-name>

**JNDI name of the EJB**

The BP Server uses the jndi name to locate the EJB in the application server, without prefixing any jndi context string, such as ejb/ or java:comp:/ejb. You must ensure that the correct jndi name is specified.

If the jndi name is different from the fully qualified class name of the home interface of the EJB, then you must specify the fully qualified class name of the home interface of the remote EP in oebpsjndi.properties file, located in OEBPS_HOME\conf folder, by adding a new property:

map.<jndi-name of EJB>=<fully qualified class name of EJB home interface>

Use this jndi name in the performer string as <jndi name of EJB>.

**Method name of the EJB**

The BP Server uses the name of the method in the EJB remote interface, to get the performer(s) for the Activity workstep. Use this method name in the performer string as <method name>.

The method signatures in the EJB should be:

public Vector <method name> (HashMap)

The Vector returned by the method should be a Vector of string objects for use as the performer(s) in the workstep.
Client view of the EJB

To connect to the EJB, the client view of EJB is required in the application server classpath. The client view typically includes the home and remote interface classes, their sub classes, Java object classes that are passed or returned from the home/remote methods, and user-defined exception classes. It is up to you to make these classes available in the application server classpath. The procedure for updating the classpath depends on your application server brand. Refer to your application server documentation for more information.

One way you can place the client view of the EJB is in the OEBPS_HOME\ebmsapps\ejbadapters folder, where ebmsapps is the application folder where all the BP Server applications are located.
E-mail templates for task completion

In Business Process Server, task completion via e-mail option can be enabled for any human workstep. However, BP Server uses fixed e-mail content and format to pass the dataslots and other process information, and also expects the data in the "reply" e-mail in a fixed format. This causes severe limitations for applications that heavily depend on the task completion via e-mail.

Note: E-mail Templates are supported only for the completion of work items or to assign the work items from available pool.

For details, see the following topics:

• Enabling task completion by e-mail
• Variables
• Expressions
• Variables using formal notation
• System variables
• User defined dataslot values
• Pre-populated output dataslot values
• API for sending e-mails to assigned task
• E-mail recipients
• E-mail subject
• E-mail backup
Enabling task completion by e-mail

To enable task completion by e-mail:

1. You must start the E-mail Reader at BP Server startup, by setting the following property in `OEBPS_HOME/conf/bpserver.conf`:
   ```
   bpserver.email.reader.start=true
   ```

2. If the BP Server is running, then you must stop and restart it for this setting to be effective. Alternatively, while the BP Server is running, you can enable the E-mail Listener by executing the following command from the BP Server Admin utility:
   ```
   startEmailListener
   ```

Variables

In e-mail templates, BP Server uses two types of variables; variables that can refer to user defined dataslots and system dataslots defined for a process template. These variables can get their values at runtime either from a process instance or process template.

A variable or identifier name is always enclosed in curly braces `{}`. To read the value of a variable, the variable name enclosed in curly braces should also be prefixed with a "$" symbol. A variable or identifier name should be either a valid user defined dataslot name or one of the specified system variables. For more information about the list of system variables that can be used with e-mail templates, see BP Server variables on page 162.

Formal reference notation

A Formal Reference Notation variable also begins with $ but the variable name should be enclosed in the curly braces `{}`. For example: `${customer}.payments were received on time.`
Formal notation usage

BP Server uses only formal notation to refer to the e-mail variables. For any value assignment to variables, variables enclosed in {} should be used in the expression without prefixing "$" symbol. However, to read the value of variables, "$" symbol should be prefixed with the variable name that is enclosed in {}.

${name}, ${price}, ${customer} // formal notation

Value Assignment

{name} = [BP Server] // to change the value of the dataslot "name".

To read the value of a variable

${name}
${customer}.payments were received on time

A system dataslot name is always prefixed with "PCTX." keyword for the proper name space use. The keyword "PCTX" is an abbreviation of term "Process Context".

System Variables

${PCTX.INSTANCE_NAME}

Expressions

BP Server uses the following expressions to access the user defined dataslot name and value, and system dataslot name and value.

List of expressions used

An <identifier> can be either a user defined dataslot name or system dataslot name.

$ {<identifier>}

Replace the entire ${<identifier>} with the value of the identifier. A reference variable can be specified any number of times and at any place within the e-mail content.

{<identifier>} = (${identifier} )

Value of the identifier is written within the parenthesis ( ). There can be only one expression per line. However, the value may exceed more than one line. These types of expressions are used only for input dataslots. Any change to the value specified in () is ignored in the reply e-mail. This is optional.

{<identifier>} = [ ]
{<identifier>} = [ ${<identifier>}]

Value of the identifier is written within the square brackets [ ]. There can be only one expression per line. However, the value may exceed more than one line. These types of expressions are used only for output dataslots. In the reply e-mail, e-mail parser reads the values within square brackets and update the corresponding dataslot in BP Server.

An identifier can be specified multiple times in an e-mail template. However, if any output dataslot is used multiple times, then the last occurrence is used as the dataslot value.
Variables using formal notation

A reference variable can be specified using the formal notation $\{dsName\}$. A user can specify any of the input or output dataslot with this notation in any place. E-mail parser replaces all dataslots enclosed in {} with the corresponding value. In general, $\{dsName\}$ notation can be used to refer the value and it can be used at any place.

"Please make the payment for the invoice $\{INV_ID\}$ dated as $\{INV_DATE\}$ sent to you few weeks before"

Note: $INV_ID$ and $INV_DATE$ are dataslots. E-mail parser replaces all occurrences of dataslot values enclosed in {} in any part of the e-mail content with the corresponding values at the time of activation of the workstep.

After parsing, the above sentence looks as follows:

"Please make the payment of the invoice "12345" dated as 03/Mar/2005 sent to you few weeks before"

System variables

All system variables should always be in uppercase and they are either prefixed with "PCTX." or "EMAIL."

BP Server variables

All BP Server system dataslots are prefixed with "PCTX.".

System dataslots supported by BP Server

BP Server supports the system dataslots explained in Table 15: BP Server variables on page 162 in e-mail templates.

Table 15: BP Server variables

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCTX.WI_ID</td>
<td>Work item ID</td>
</tr>
<tr>
<td>PCTX.ENC.WI_ID</td>
<td>Encrypted work item Id</td>
</tr>
<tr>
<td>PCTX.TEMPLATE_ID</td>
<td>Process template Id</td>
</tr>
<tr>
<td>PCTX.INSTANCE_ID</td>
<td>Process instance Id</td>
</tr>
<tr>
<td>PCTX.WS_ID</td>
<td>Workstep Id</td>
</tr>
<tr>
<td>PCTX.TEMPLATE_NAME</td>
<td>Template name</td>
</tr>
<tr>
<td>PCTX.INSTANCE_NAME</td>
<td>Instance name</td>
</tr>
</tbody>
</table>
When it is required to specify the system datatask in the e-mail content, you can specify the system identifier enclosed in {}. As with any reference variables, e-mail parser replaces all system identifiers enclosed in {} with their proper values.

Please complete the task ${PCTX.WORKITEM_ID} asap.

E-mail variables

All e-mail variables are prefixed with "EMAIL.".

E-mail variables used by BP Server

BP Server uses the e-mail variables explained in Table 16: E-mail variables on page 163 that can be specified at any place in the e-mail content.

Table 16: E-mail variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Identifier</th>
<th>Description</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>CC</td>
<td>EMAIL.CC</td>
<td>Comma separated e-mail IDs</td>
<td>User</td>
</tr>
<tr>
<td>BCC</td>
<td>EMAIL.BCC</td>
<td>Comma separated e-mail IDs</td>
<td>User</td>
</tr>
<tr>
<td>Email Subject</td>
<td>EMAIL.SUBJECT</td>
<td>May contain system and user datatask</td>
<td>User</td>
</tr>
<tr>
<td>Email User</td>
<td>EMAIL.USER</td>
<td>Performer name</td>
<td>User</td>
</tr>
<tr>
<td>Security Key</td>
<td>EMAIL.SKEY</td>
<td>Encrypted value of work item ID</td>
<td>System</td>
</tr>
</tbody>
</table>

---

3 This variable is used only by the text e-mail template.
4 This variable is used only by the HTML e-mail template.
### User defined dataslot values

**Input dataslots**

An e-mail template may optionally specify the input dataslot values as below:

```
(dsName1) = (${dsName1})
(dsName2) = (${dsName2})
(dsName3) = (${dsName3})
```

**Output dataslots**

The output dataslot values may be specified as below:

```
(dsName5) = []
(dsName6) = [] (choices: c1, c2, c3)
(dsName7) = []
```

If there are any choices for the output dataslots, then they are always appended after the closing square bracket "]" as "(choices: c1, c2, c3)".

### Pre-populated output dataslot values

The BP Server variable `PCTX.OUTPUT_DATASLOTS` represents the list of output dataslots for a workstep in a text e-mail template, whereas the variable `PCTX.ALL_WS_DATASLOTS` represents the list of output dataslots for a workstep in an HTML e-mail template. However, by default, output dataslot values are printed also. The user can modify these output dataslot values while sending the reply e-mail.
**Printing of output dataslots**

If you do not want to print the values of output dataslots, then you can set the value of the property "bizlogic.email.template.print.outputvalues" to False. The default value is always true. However, this variable is required only for the BP Server variable PCTX.OUTPUT_DATASLOTS. If the user writes custom template, then it is possible to control which output dataslot value to be printed.

```java
(dsName1) = [{$dsName1}] // to show the current value of "dsName1" to e-mail user
(dsName2) = [] // current value is not shown to the e-mail user
```

**API for sending e-mails to assigned task**

If the e-mail task user lost the e-mail received from Business Process Server, then there is no way to complete the task using e-mail. Also, there is no way to resend an e-mail for an already created and assigned work item. In such cases, the following BP Server API can be used to resend an e-mail for any work item in an Assigned state.

```java
WorkitemSVO: public boolean sendAssignedTask()
```

**E-mail recipients**

Like any other e-mail, the e-mail for task completion is delivered to the recipients mentioned in the To, CC and BCC fields. This task is assigned only to the To recipient(s). Though the task is not assigned to the CC and BCC recipients, any one of them can complete the task or assign it. However, when a CC or BCC recipient completes or assigns a task, a warning to that effect is recorded in the bpserver.log file.

**E-mail subject**

Currently, BP Server constructs the e-mail subject as follows:

**Format:**

- **Assigned WorkItem:** `<e-mail subject prefix>Assigned Task <wsName (wiid)>`
- **Available WorkItem:** `<e-mail subject prefix>Available Task <wsName (wiid)>`

where `<e-mail subject prefix>` is configurable using the `bizlogic.email.subject.pattern` parameter in the `oebpsemail.properties` file. By default, the value of this parameter is `bpserver#`.

```plaintext
Assigned WorkItem: sbm#Assigned Task <Activity 1 (24)>
Available WorkItem: sbm# Available Task <Activity 1 (24)>
```

If subject is not provided by the user in the case of custom e-mail templates, then BP Server constructs the default subject in the same format as mentioned above.
E-mail backup

Currently, the e-mail reader continuously polls on the specified inbox. All BP Server e-mails for task completion are processed and removed from the folder. There is no support to backup the successfully processed e-mails.

With IMAP protocol, it is possible to create backup folders. However, there is no such support for POP3. BP Server moves all deleted e-mails to a backup folder. User can specify the name of the backup folder using the following property in the `oebpsemail.properties`:

```
bpserver.email.backup.folder
```

This property can be used only with IMAP protocol and not with POP3 protocol.

E-mail search

BP Server e-mail reader reads only BP Server e-mails and leaves all other unknown e-mails in the INBOX intact. This is possible by reading e-mails whose subject contain a specific keyword like “BP Server”. Java Mail API supports a feature to define search criteria to download e-mails that satisfy the given conditions.

Subject pattern for e-mails

The default subject pattern to download only Business Process Server e-mails is specified as `bpserver#`. This means, the e-mail reader reads all e-mails whose subject begins with `oebps#`. User can modify this default pattern to some other value using the following property:

```
bpserver.email.subject.pattern=oebps#
```

However, please note that this pattern must not contain any spaces. If the user prefers no prefix for the custom messages, then user may remove this default value from the property file. This approach may prove dangerous if the user specifies his personal e-mail id for inbox polling, because all non-BP Server e-mails are deleted from the folder as there is no prefix to identify valid BP Server e-mails.

E-mail clients

It is difficult to support all e-mail clients because they are either platform-specific or their e-mail content format is specific to the local settings. For example, some e-mail clients add different special characters to fold long lines or write different EOF or EOS characters for the end of e-mail.

Properties used to handle variations in dataslot values

When CHARACTER dataslot value is specified in the e-mail body, some e-mail clients fold the string value into multiple lines separated with “\n” or “\n\n” depending on the platform. Therefore, BP Server uses the following property to handle some of the variations of these e-mail clients:

```
bpserver.email.remove.chars.l=\r\n
```

User can modify the above property for other special characters.

To configure your e-mail client and enable the JavaScript option to use the task completion by e-mail feature, refer to your e-mail client's documentation.
E-mail security

BP Server uses Business Process Server’s password encryption and decryption utility PService for the secured e-mail task completions.

Workflow completion

When BP Server sends an e-mail for a work item completion, it uses the PService utility to encrypt the work item ID that is included in the e-mail content. The encrypted work item ID is decrypted when the reply e-mail is received. If the decrypted work item ID is invalid, then a reply e-mail is sent and also proper error messages is written to the log file.

For each e-mail that BP Server sends, it appends the security key at the end of the e-mail. The security key is the encrypted value of the work item ID. A reply e-mail without a valid encrypted key cannot be processed and is sent to the trash folder.

{EMAIL.SKEY} = [123456789]

When a reply e-mail is received and picked up for processing, the parser first reads the encrypted security key, decrypt its value using PService utility and also validate with BP Server. If any one of these operation fails, then the e-mail is ignored and the sender is notified with an error message.

Types of e-mail templates

Frequently, applications require more meaningful e-mail content with datslot values that give complete context of the process data to the receiver. In such cases, e-mail templates provide the easiest, simplest, and cleanest way to incorporate dynamic content for e-mail task completion. E-mail Templates provide a secured free content format that can be specified for any workstep enabled for task completion using e-mail option.

You can use either the text or HTML e-mail templates. In each of these template type, you can create templates with dynamic content, custom template or use the default template. The order in which BP Server searches for these templates and other details about them are explained in Table 17: E-mail Templates on page 167.

Table 17: E-mail Templates

<table>
<thead>
<tr>
<th>Template</th>
<th>File name</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>E-mail Template with Dynamic Content</td>
<td>&lt;filename&gt;_assigned.html &lt;filename&gt;_available.html</td>
<td>OEBPS_HOME\ebmsapps&lt;appName&gt;\templates\email&lt;wsName&gt;\</td>
</tr>
<tr>
<td>Custom E-mail Template</td>
<td>assignedwi.html availablewi.html</td>
<td>OEBPS_HOME\ebmsapps&lt;appName&gt;\templates\email&lt;wsName&gt;\</td>
</tr>
<tr>
<td>Default Template</td>
<td>assigned.html available.html</td>
<td>OEBPS_HOME\ebmsapps\common\templates\email\</td>
</tr>
</tbody>
</table>

5 The extensions for the file names shown in the table are for the HTML templates. For Text templates, the extension is .etl.
Text e-mail template

The Text e-mail templates are meant to provide the easiest, simplest, and cleanest way to incorporate dynamic content for the task completion by e-mail. It provides a secured free content format that can be specified for any workstep enabled for task completion using e-mail option.

**Note:** Certain e-mail clients indent the original text with some special characters like “>” or “[“ while replying. These extra characters added by the e-mail client must be removed manually before sending the reply e-mail.

E-mail templates locations and file format

Location of default templates:

%OEBPS_HOME%\ebmsapps\common\templates\email\n
User defined custom templates:

%OEBPS_HOME%\ebmsapps\<ptName>\templates\email\<wsName>\n
Template file name format

Assigned WorkItems: assignedwi.etl
Available WorkItems: availablewi.etl

The suffix "etl" stands for "E-mail Template Language". When there is no custom template defined for a workstep, then BP Server constructs the e-mail content using the default template. If you change the default templates’ file names, then no mails are sent for task completion. Therefore, it is recommended that you should not change the default templates’ file names.

The default templates assignedwi.etl and availablewi.etl are used to construct the e-mail when there are no custom templates defined for a workstep.

Sample e-mail template for a workstep

*****************************************************************

Dear admin

We would like to inform you that a new task ${PCTX.WORKITEM_ID} is assigned to you. The product ${PROD_ID} with the description ${PROD_DESC} found to have less inventory. So we would like to place the order ${ORD_QTY} for the mentioned product to meet the customer requirements.

However, please let us know your requirements for the mentioned product with proper details so that we can place one order for the combined quantity. Please enter the required values within the square brackets.

Input DataSlots:
{PROD_ID} = ( ${PROD_ID} )
$PROD_DESC} = ( ${PROD_ID} )

Output DataSlots:
{ORD_QTY} = [ ]
{SPECIFICATION} = [ ${SPECIFICATION} ]
{REQ_DATE} = [ ]

Thanks and Regards
BP Server default e-mail templates

The default template files assignedwi.etl and availablewi.etl are used for all worksteps for which no custom templates are specified. Users can also modify the default templates at %OEBPS_HOME%/ebmsapps/common/templates/email as per their requirements.

Default template for assigned work items
assignedwi.etl:

Dear ${PCTX.WI_PERFORMER}

The details of the workitem are printed below.
Application: ${PCTX.TEMPLATE_NAME}
Instance: ${PCTX.INSTANCE_ID}
WorkStep: ${PCTX.WS_NAME}
WorkItem: ${PCTX.WI_ID}

The following input dataslots should not be modified and any change to the value of these dataslots is ignored. ${PCTX.INPUT_DATASLOTS}

Please enter the value for all output dataslots within the square brackets. ${PCTX.OUTPUT_DATASLOTS}

Thanks and Regards
BP Server Administrator

To complete the task, you may either reply to this e-mail or go to the URL. The URL option works only if you have the access to the firewall. ${PCTX.BIZSITE_LOGIN_URL}

$EMAIL.SUBJECT = Assigned Task <${PCTX.WS_NAME} (${PCTX.WI_ID})>

Default template for available work items
availablewi.etl:

Dear ${EMAIL.USER}

A workitem is made available to you. The details of the workitem are printed below. Please reply to this e-mail to assign the workitem to yourself.

Application: ${PCTX.TEMPLATE_NAME}
Instance: ${PCTX.INSTANCE_ID}
WorkStep: ${PCTX.WS_NAME}
WorkItem: ${PCTX.WI_ID}

Thanks and Regards
BP Server Administrator

$EMAIL.SUBJECT = Available Task <${PCTX.WS_NAME} (${PCTX.WI_ID})>
Custom text e-mail template

The file name of this type of template should be the same as the default text e-mail template file name (assignedwi.etl or availablewi.etl). These custom text e-mail templates are stored in the %OEBPS_HOME%/ebmsapps\<ptName>\templates\email\<wsName>\ folder. If this template is present, then it is given higher precedence compared to the default template file.

Additional operations

In addition to displaying dataslot values, you can add a text in the custom template for various purposes like giving additional information about the tasks or explaining instructions to be followed. However, please note that these custom e-mail templates give you very limited formatting options.

E-mail template viewer

One way of testing and verifying the template files is to use it with a process template. But this requires a lot of time and effort in setting up the e-mail server, e-mail account, creating the test application etcetera.

TemplateViewer

A command-line utility TemplateViewer can be used to verify the default or custom templates. This utility creates an instance of the template using the input and output dataslots of the specified work item and writes the entire instance to a file. Using this utility you can fix any errors or format issues of the template before publishing it for the production. This utility TemplateViewer.sh/.cmd is available in %OEBPS_HOME%\bin.

Usage:

```
TemplateViewer [OPTIONS]
OPTIONS:
  -u: Username to connect to the BP Server
  -p: Password to connect to the BP Server
  -wi: Workitem ID
  -f: [OPTIONAL] File name to write the template instance.
```

```
TemplateViewer.cmd -u ebms -p ebms -wi 12345 -f c:\oebps_home\mytemplate.txt
```

HTML e-mail templates

Most of the limitations of the text e-mail templates can be overcome by using the HTML e-mail templates.

Features of the HTML e-mail template

The HTML e-mail template provides you the portal functionality and ease of use. Work items are pushed to the performers with portal views that can be customized for every task. HTML e-mail templates support client-side and server-side mandatory dataslot validation. This validation is controlled by the bpserver.email.required.dataslots.check parameter in the oebpsemail.properties file. By default, this validation is enabled. Similarly, client-side data validation can also be done by using the inline JavaScript. The default HTML template supports all data types except MAP and OBJECT dataslots.
HTML e-mail supports inline stylesheets for formatting. It also allows you to embed images by using relative path, file URL, or http protocol. You can customize the default HTML generator, header and footer of the HTML e-mail template. It supports reusable JavaScript code for custom HTML templates.

The e-mail task performer does not need a separate authentication. It uses single sign-on to inbox and to complete the tasks. HTML e-mail template does not pose any authentication issues and it is not impeded by a firewall.

For your convenience, HTML e-mail template provides a main help and context help for complex dataslots. However, you may modify the mail help as per your requirement, while creating a custom e-mail template.

**SendHtmlEmail method**

The SendHtmlEmail method extends text based SendEmail method and provides the following additional services:

- setContent(String htmlContent)
- addHtmlImage(URL url, String contentId)
- setContent(URL htmlPage, Hashtable images)

You can extend the SendHtmlEmail() method to customize the message parts by overriding the assembleMessage() method.

**SendEmail authentication**

You can use the following parameters in the oebpsemail.properties file and set the user name and password to enable the sendemail authentication in BP Server.

- oebps.email.outgoing.server.user
- oebps.email.outgoing.server.password

If the user name and password values are blank or null, then no authentication check is performed. Applications can make use of the new constructor of the SendEmail class to send an e-mail with authentication.

**HTML images**

Images specified through `<img>` tag are embedded in the HTML e-mail template with a unique content-ID. Images can be referred using a relative path like `<IMG SRC="images/flower.jpg">`, a file URL like `<IMG SRC="file:///c:/htmlemails/images/flower.jpg">` or the http protocol like `<IMG SRC="http://www.sender-company.com/images/flower.jpg">`. An image referred to by the http protocol is not embedded in the e-mail. When images are referred by relative path, or file URL, the SendHtmlEmail method reads the image file's binary content and adds it to the e-mail container. It creates a new MimeBodyPart for each image.

**Content-ID URLs for images**

Sometimes, the e-mail server cannot resolve the embedded images uniquely for the following reasons:

- Same image file name with different content may be referred to at multiple places in a single HTML page.
- Same image file name with different content may be referred to from multiple e-mails.
You can overcome this issue by using the content-ID URLs for the images. These content-ID URLs resolve the images uniquely for each image referred to in each e-mail and within each e-mail content. If the text of an HTML e-mail contains a link to other part of the same e-mail, then it should also use the content-ID URLs to refer to another part of the same e-mail. The content-ID URLs are case-sensitive. You can use the protocol "cid:" to define a content-ID URL. Content-ID URL of a specific length can be generated by using the CIDGenerator class. To ensure uniqueness across e-mails, BP Server uses the following content-ID URL format:

\(<10\text{-digit random number}>+\text{outgoing-mail-server+.img}\) 

When a content-ID ends with .com, the e-mail server may assume that the attachment is a binary executable and reject the e-mail. To avoid this issue, .img suffix is added. It also conveys that this content-ID is generated for an <img> tag.

**HTML parser**

BP Server uses an HTML Parser to parse the HTML for all <img> tags and automatically replace the relative or file paths of the images with uniquely generated content-ID URLs. For example, the <img src="images/loginlogo.gif"> tag is replaced with the <img src="cid:{unique-id}"> tag.

The HTML Parser can also be used to plug-in the generated HTML into an HTML template, similar to the Dynamic HTML. BP Server needs a well formed HTML e-mail template, otherwise it cannot embed images with sendemail.

**Multipart e-mail content**

An HTML e-mail's content can be divided into multiple parts like plain text, HTML, image. The structure of an HTML e-mail with embedded image but without a document attachment is shown in Figure 17: HTML e-mail with images but no document attachment on page 172.

**Figure 17: HTML e-mail with images but no document attachment**
The structure of an HTML e-mail with a document attachment is shown in Figure 18: HTML e-mail with document attachments on page 173.

**Figure 18: HTML e-mail with document attachments**

![HTML e-mail with document attachments](image)

### HTML template with dynamic content

An HTML e-mail template file comprises of a header, the process-data, and a footer, and it has an ".htl" extension. An HTML e-mail template can use any of the e-mail variables and/or process-context variables. To complete a task successfully using an HTML template, you must enable the JavaScript option in your e-mail client. Otherwise, when you click the Submit button in your e-mail client, an error message is displayed and the task is not completed.

HTML templates can be classified into the following categories:

- **Default templates** (assigned.HTL, available.HTL)
- **Custom templates with static file names** (assignedwi.HTL , availablewi.HTL)
- **Custom template using form fields**

A custom template using form fields can be specified through the FormName field of the workstep properties. The file name for a custom template using form fields can be specified through a mapped dataslot and it must end with "_assigned.HTL" or "_available.HTL".

The directories where these templates are stored are as follows:

- **Custom template using form fields**: `OEBPS_HOME\ebmsapps\<appName>\templates\email\<wsName>`
- **Custom Templates with static file names**: `OEBPS_HOME\ebmsapps\<appName>\templates\email\<wsName>`
- **Default Templates**: `OEBPS_HOME\ebmsapps\common\templates\email`
Also note that in the case of a custom template using form fields, the mapped dataslot template filenames should end with ".assigned", whereas in the case of a custom template with static filename, the filenames should end with "assignedwi" or "availablewi".

The order in which BP Server searches the templates is as follows:

- Custom template using form fields
- Custom Templates with static file names
- Default Templates

The Header(header.h1) and Footer/footer.html) templates from the \common\templates\email folder are included in all type of templates mentioned above.

Depending on the template type specified for the bpserver.email.preferred.template.type parameter in the oebpsemail.properties file, BP Server selects either etl or htl templates. If the specified template is not available at the specified location, then BP Server looks for any other template in the same location.

**Default HTML template**

The default HTML template supports all data types except the Map and Object data types. It uses labels defined for the dataslots.

**Enabling HTML templates**

To enable HTML templates, set the bpserver.email.preferred.template.type parameter in the oebpsemail.property file to "htl".

**Custom HTML e-mail template**

This section explains how to use the custom HTML e-mail templates.

**Custom templates with static file name**

The file name of this type of template should be the same as the default template file name (assignedwi.h1 or availablewi.html). If this template is present, then it is given higher precedence compared to the default template file.

**Custom templates using form fields**

For this type of template, the file name can be specified dynamically by using the "FormName" field of the Workstep Properties.

**Setting values withing fields**

The value of the "FormName" field can be a file name ending with ".assigned.h1", ".assigned.etl" or a mapped dataslot name. In the case of mapped dataslots, BP Server gets the template file name from the mapped dataslot at runtime. If you specify a file name other than the recommended ones (even in for the mapped dataslots), then it is ignored.

If a workstep is assigned to user(s), group(s) or queue(s), then assigned task e-mails are sent using only the ".assigned" template file name.
If a workstep is available to user(s), group(s) or queue(s), then e-mails for both, available task and assigned tasks, are sent. The assigned tasks e-mail is generated using the "_assigned" template file name. While generating an e-mail for an available task, BP Server looks for a template having the same name as that of the assigned template except that the string "_assigned" is replaced with "_available", in the same location. If such a file is found, then BP Server uses that template for sending the available task e-mail. If such a file is not found, then a check is made for a custom available template; if that also is not found, then it uses the default available template.

**Configuring a custom template using the form fields**

**To configure a custom template:**

1. In the Progress Developer Studio for OpenEdge, open the Workstep Properties window.
2. Click the **General** tab.
3. On the **Presentation** section, click Add to open the **Presentation Types** window.
4. From the Category drop-down list, select Portal.
5. Do one of the following:
   - If you want to specify a template name, then select Custom from the **Type** drop-down list and enter a template file name in the **Name** text box.
If you want to specify a mapped dataslot, then select Custom from the Type drop-down list and select the dataslot to which it must be mapped from the drop-down list.

6. Click OK.

The structures of the default and custom template is displayed in the Figure 19: Default and custom template structure on page 176.

Figure 19: Default and custom template structure

HTML composer

The default HTMLComposerImpl class generates the HTML code on the fly for each work item that gives HTML view for the work item data including the table view for input and output dataslots. It generates editable fields for the output dataslots and disabled fields for the input dataslots. While generating the HTML code, the order of input and output dataslots is maintained as specified when the application was designed. It supports all data types except the Map and Object types. You can use the HTML special characters like <, >, ",," as a part of the dataslot values. BP Server encodes and decodes such special characters correctly. You must define the sbmDataForm form and invoke the JavaScript function completeEmailTask() on submit of this form.

The abstract HTMLComposer class provides following services:

- Getter methods for all dataslot meta-data
  
  public abstract String create(boolean editableFields);

- Public abstract String create();

The HTMLComposerImpl class provides the following services:

- createReadOnlyFields()
Guidelines for custom template

The following section provides the guidelines for a custom template.

Guidelines

• The header.htm must be inserted after the <head> tag.
• The footer.htm must be inserted before the </body> tag.
• Define the form with the name sbmDataForm and invoke the JavaScript completeEmailTask() on submit of this form. For example, <input type="button" name="completeTask" value="Complete Task" onClick="completeEmailTask();">
• All input fields should be defined with a “disabled” attribute. For example, <input name='dsName' type='text' value='${dsName}' disabled>
• JavaScript completeEmailTask() must be defined in the header.htm and this function and all dependent functions should not be modified.
• You may add new reusable JavaScript functions and Stylesheets that are common to all templates in the header.htm.
• Any application specific JavaScript and/or Stylesheets should be added only to individual custom templates and not to the header.htm.
• Custom templates should use well formed HTML.
• E-mail clients do not resolve HTML tags with reference to external files using file protocol or relative/absolute path. However, they may work fine if the references are using http protocol.
• If the custom template’s application specific stylesheet conflicts with the stylesheet in the header.htm, then you can define the application specific stylesheet inside the <body> tag.

An example of custom HTML template

<html>
<head>
<!- header.htm is inserted here ---? 
<title>Product Selection</title>
</head>
<body><br/>
<table border="1" cellspacing="1" cellpadding="1">
<tr>
Reply e-mail

Task completion by e-mail is performed by filling up the output dataslot values of the `sbmDataForm` form. On submittal of this form, the `completeEmailTask()` function compose a plain text with all output dataslot values and process context, and sets this text as the body content for the `mailto:` tag. Therefore, to work this feature correctly, you should not modify the content of the reply mail. The reply e-mail is in plain text format as required by the BP Server EmailReader. If you are using the Netscape e-mail client, then you must disable the HTML e-mail composition option.

Decoding of document file names

JavaMail automatically encodes the e-mail message parts before sending an e-mail and after receiving it, decodes the message while reading it. The MIME specification does not allow header parameter values such as attachment’s file name to be encoded. That is why JavaMail does not encode or decode filenames in the header. Business Process Server's EmailReader is configured to decode the filename in the order specified below:

- Base64
- QuotedPrintable

EmailReader uses JavaMail’s MimeUtility and Commons-Codec to decode the encoded file names.

Features of default template

- The default template provides a GUI to display all the dataslots except MAP and OBJECT dataslots.
- Help is provided for all e-mails generated using the default template.
• For some of the complex dataslots, context-sensitive help is provided, which can be accessed by moving the mouse over the info area located on the right side of the dataslot value. Such a help is provided for the List and DATETIMETZ dataslots.

• The default template can handle complex dataslots. Depending on the type of your dataslot, follow the instructions given below to fill the data for complex dataslots:
  
  • **LIST** — For a LIST dataslot, the items should be added in the list. Only the selected items are part of the reply e-mail. The reply e-mail is generated after clicking the Complete button. To delete an item, please deselect it from the list. To deselect an item in the list, press the CTRL button and deselect the item.
  
  • **DATETIMETZ** — For a DATETIMETZ dataslot, the required format date value is provided in the context help. The format of the date value depends on the value of the following two parameters in the oebps.conf file: oebps.dateformat.date= and oebps.dateformat.time=

### Dataslot handling

#### Text E-mail Templates

• No support for client-side data validation.
• No support for client scripting.
• Difficult to format multiple choice dataslots, collection dataslots and other complex types.
• No standard GUI tool to create text-based e-mail templates.
• No support for images.
• Difficult to extend the template beyond the customization of the text format of the e-mail content.

#### HTML E-mail Templates

• Only images can be embedded with HTML e-mails.
• No support for references to external files in any other HTML tags.
• E-mail clients do not allow the following functions of the "window" object:
  
  • Window.focus()
  • "Window.name"
  • "Window.opener" should NOT be used because it is not initialized with the reference of the parent window.

• BMP files are not displayed properly in the Netscape.
• Only selected e-mail clients provide complete support for HTML e-mails.
• JavaScript must be enabled for e-mail task completion.
• With Netscape, HTML e-mail composition must be disabled.
• When it is required to create a pop-window, then `window.open()` should be used for Netscape and `createPopWindow()` should be used for OutLook and Thunderbird.

• OutLook and OutLookExpress encode spaces in a CHARACTER dataslot value to "+". To solve this issue, set the `bpserver.email.decodeplus.outvalues` parameter in the `oebpsemail.properties` file to true. This converts all the "+" signs to space. This should be used carefully because any valid "+" is also converted to spaces.

• With Outlook, the total byte size of the composed e-mail should be less than 1445 bytes. Otherwise, Outlook either displays an error or does not respond.
Inline adapters

BP Server provides adapters as a mechanism to execute external Java class as a part of the Business Process. For an Adapter workstep, you must specify a fully qualified java class name, the input and output datalot names, and the method name to be invoked. Once such a process instance is created and the execution reaches the Adapter workstep, the setter methods on the adapter are invoked. These methods set the input datalot values, invoke the specified adapter method and then invoke the getter methods to get the output datalot values.

For details, see the following topics:

- Inline adapters

Inline adapters

The adapter implementation typically uses Java Messaging Service (JMS) APIs and infrastructure. When an adapter is invoked, a JMS message gets sent to a JMS Queue. A Message Driven Bean (MDB) listening to this JMS Queue picks up the message, decodes it and executes the adapter. This method is inefficient and slow. Therefore, BP Server provides an option to overcome this limitation. If you select this option, then the adapter is executed in the same thread instead of using JMS infrastructure for adapter invocation.

Usage of Inline adapters

While designing a process template in the Progress Developer Studio for OpenEdge, you can select the option for an Adapter workstep to execute the adapter in same thread. This sets the ‘inline’ flag for the adapter in the process template’s XML file.
After creating a process instance of such a process template with an inline adapter, when the execution thread reaches adapter execution, BP Server detects the inline adapter. Consequently, it directly executes the adapter in the same thread instead of sending a JMS message. Thus, the adapter is executed using the same thread that completed the previous workstep.

**Important points**

- Only an additional property is introduced for the Adapter workstep. There is no change in the adapter development. The adapter structure remains same.

- If a process template has all its worksteps as inline Adapter worksteps, then the entire process instance is completed in the same thread of execution. Thus, all adapters are executed in the same thread.

- Inline adapters should not start any EJB transaction.

- Any datasource used in an inline adapter code should be an XA enabled datasource.

- Only local synchronized adapters can be configured as inline adapters. All other adapters, like `ejbadapters`, should not be configured as inline adapters.

- Inline adapters are recommended only for short lived adapters. The following issues may occur with the long running inline adapters:
  - Transaction timeout may happen, and hence, it may cancel completion of previous workstep.
  - It may lock the system resources, like thread, and application resources, like database connections, cursors etcetera. Such locking may affect the performance, maximum number of concurrent users/access, and may cause other resource issues.

**Limitations**

Inline adapters are always executed using EJB transactions. In contrast, non-inline adapters, which are executed via an MDB, are never involved in a transaction.
JMS based event publisher

Business Process Server uses Java Message Service (JMS) for publishing the BP Server events to the external applications. This chapter describes the JMS based Event Publisher.

With each state change of the business process, BP Server generates an event. These generated events asynchronously communicate with the external applications.

For details, see the following topics:

• Event publisher architecture
• Event reader
• Event sender
• Event channels
• Event filters
• Event publisher example
• Administering the event publisher
Event publisher architecture

The architecture is shown in Figure 20: JMS event publisher architecture on page 184.

Figure 20: JMS event publisher architecture

The BP Server saves all the events in the BizEvent Table. The EventReader polls on the BizEvent table and reads all the events periodically. You can modify the configuration file of the EventReader to define the number of events to be read for each database access, the transaction size, and the sleep interval between database accesses.

The EventReader reads the events from the database, and invokes the publish(CommonEvent) method on each sender class. The sender class applies the prefilter conditions on the event and then publishes using JMS or any other technology. The EventPublisher supports a default sender class implemented using JMS. You can also specify any other sender classes in the property file.

An event channel is essentially a placeholder for the specific type of event from where the subscribers or listeners retrieve the events. For the default JMS sender, the event channel corresponds to a JMS topic or queue. The type of events sent to an event channel are controlled by using prefilter properties. For example, you can define separate event channels for each process template, so that events of a specific process template are published to its specific event channel.

The default JMSSender class reads the event channel information from the database and caches it into memory. You can apply the post-filter conditions on the event channel to control the message delivery to the listeners or subscribers. In the case of JMS, you can choose the JMS SQL92 queries as post-filter conditions by including the process data in each JMS message as message properties. The JMS Server tests these conditions and delivers the message to a listener only when the SQL92 query returns true.
Event reader

The event reader reads all the events from the database periodically, and passes them to all the sender classes specified in the configuration file, by invoking the `publish()` method. The event reader logic is shown in Figure 21: Event reader on page 185.

Figure 21: Event reader

```
Event
   ↓
Senders List
      (class names specified in property file)
    ↙
   getSender
      ↓
     Yes
    ↘
Invokes
Sender.publish(Event)
      No
   ↘
Repeat for next event
```

The property file `bpserver.conf` in `OEBPS_HOME\conf` defines the properties explained in Table 18: Event reader properties on page 185 to control the EventReader.

Table 18: Event reader properties

<table>
<thead>
<tr>
<th>Property name</th>
<th>Description</th>
<th>Default values</th>
</tr>
</thead>
<tbody>
<tr>
<td>publisher.event.batch.size</td>
<td>Number of events to read in a single database access.</td>
<td>200</td>
</tr>
<tr>
<td>publisher.event.transaction.size</td>
<td>Number of events to process and publish in a single database transaction.</td>
<td>10</td>
</tr>
<tr>
<td>publisher.sleep.time</td>
<td>Sleep time between each database access</td>
<td>10000 ms</td>
</tr>
<tr>
<td>publisher.senders.list</td>
<td>Comma separated list of fully qualified sender classes. By default, JMSSender class is specified.</td>
<td>com.savvion.sbm.event publisher.sender.JMSSender</td>
</tr>
</tbody>
</table>
Optionally, advanced users can add the properties explained in Table 19: EventReader optional properties on page 186 to the `bpserver.conf` in `OEBPS_HOME\conf` file. Note that these properties must be explicitly specified.

### Table 19: EventReader optional properties

<table>
<thead>
<tr>
<th>Property name</th>
<th>Description</th>
<th>Valid values</th>
</tr>
</thead>
<tbody>
<tr>
<td>publisher.event.type</td>
<td>By default an object of <code>com.savvion.sbm.util.CommonEvent</code> is sent to the EventChannels defined. If required, the user can configure the EventPublisher to send only the context (Map) of the CommonEvent to the EventChannels defined.</td>
<td>1 (for CommonEvent) 2 (for Event Context inMap)</td>
</tr>
<tr>
<td>publisher.start.event.id</td>
<td>When this property is defined with a valid value, then upon startup EventPublisher reads all events with event ID value greater than the specified value.</td>
<td>The value could be any non-zero positive number.</td>
</tr>
</tbody>
</table>

### Event sender

The BP Server provides default implementation of the Sender interface using JMS. All sender classes should implement the Sender interface. EventReader then invokes `publish(CommonEvent)` method on each sender class specified in the configuration file.

```java
import com.savvion.sbm.util.eventreader.Sender;
public interface Sender {
    public void publish(CommonEvent evt) throws Exception;
}
```

### JMS sender implementation

This is the default sender implementation provided in BP Server. The JMS sender logic is shown in Figure 22: Event sender on page 186.

**Figure 22: Event sender**
The JMSSender reads the channel list from cache. When no filter conditions are specified for a channel, all events are unconditionally published to that channel. When filter conditions are specified for an event channel, they are first evaluated using the input event. The event is published only when the specified simple or compound conditions are true.

The Event Publisher assumes that all JMS destinations specified as channels are accessible with the default JMS server that is directly by the application server.

User-defined sender implementation

BP Server supports user-defined sender implementation. You can write a simple Java class that implements the Sender interface and add this class to the senders list specified in the property file. The EventReader passes all the events unconditionally to this class. You can specify any number of sender classes in the list.

When you write a new sender class, you can specify it as the default sender class or add it to the comma separated senders list. In the new sender class, you can handle and publish the message to a JMS or non JMS destination. The EventPublisher receives any exception raised from the sender class and initiates the shutdown process.

Business Process Server also provides utility methods to send messages directly to JMS topic or queue. You may use these utilities or write your own code to send or publish messages to JMS destinations. Use the following methods in BMJMSMessage in sender class to send messages to their destinations. The caller of these utility methods should have the following data ready.

- Destination Properties(jmsProps):
  
  ```java
  HashMap hm = new HashMap();
  hm.put(BMJMSService.DESTINATION_NAME, "myTopic"); //must
  hm.put(BMJMSService.DELIVERY_MODE, new Integer(DeliveryMode.NON_PERSISTENT)); //optional
  hm.put(BMJMSService.TIME_TO_LIVE, XXXXX); //optional
  hm.put(BMJMSService.ACKNOWLEDGE, "AUTO"); //optional
  hm.put(BMJMSService.USER_NAME, "username"); //optional
  hm.put(BMJMSService.PASSWORD, "passwd"); //optional
  ```

- To send a message to queue or topic:
  
  ```java
  BMJMSService.self().sendMessage(Map destProps, Object msg, int msgType, selectors);
  ```

  Example:

  ```java
  BMJMSService.self().sendMessage(hm, eventObj, BMJMSService.OBJECT_MESSAGE, selectors);
  ```

  The selectors are defined for post-filter conditions described in Post-filter conditions on page 191.

- To send a message to a destination in a remote JMS Server:
  
  In addition to the above-mentioned data, the sender code should also add the connection factory object to the name value pair.

  ```java
  hm.put(FACTORY_OBJECT, factoryObject);
  ```

  The factory object must be of type javax.jms.ConnectionFactory.
Event channels

An event channel describes a JMS topic or queue and associated prefilter conditions. The prefilter conditions decide the type of events to send to a particular channel. All the subscribers or listeners of this channel receive only the events filtered on the prefilter conditions. Additionally, each subscriber can also use SQL92 queries on message properties as post-filter conditions. The SQL92 based queries use the properties specified in the JMS header. The event channel architecture is shown in Figure 23: Event channels on page 188.

Figure 23: Event channels

Event channel name

The channel name must be the fully qualified JNDI name of a JMS destination. The JMS destination can be either a topic or a queue. All channel names must be unique, and duplicate entries are not allowed.

Event channel persistence

You can define event channels using Business Process Portal. For more information about defining event channels, see "Chapter 11, Managing event channels" in Business Process Portal Administrator’s Guide.

Note: BP Server does not provide any configured channel by default for publishing events. Therefore, it does not publish any event when you start EventPublisher. You can add channels for publishing events by using Event Publisher feature of Business Process Portal’s Administration module. After you configure channels for publishing events, BP Server can publish events on them.
You can also specify an expiration date for an event channel. When an expiration date is specified, the channel is removed automatically after the expiration date. When no expiration date is specified, the channel information always persists and the EventPublisher continues to publish all valid events to this channel.

**Note:** The expiration date of an event channel is effective only when the EventPublisher is started or refreshed. It is also effective when any event channel is added, modified, or deleted.

---

**Event channel cache**

For each event sent to each sender class, all event channels are accessed and the event is published only to the qualified channels. Filter conditions of each channel are evaluated using the given event object and the event is published to this channel only when the filter conditions are evaluated to true.

The channel information is cached for faster reference. The cache is created for the first time when the Event Publisher is started. Event Channel cache is refreshed whenever a new channel is added, or when a client utility is invoked to send the REFRESH message. When the cache is refreshed, Event Publisher maintains consistency by ensuring that the sender class is not using the cache.

The cache is maintained as a collection of value objects. During the start and refresh method invocation, all the channels are validated for expiration date, and the expired channels are removed from cache and database.

---

**Event channel examples**

You can create, modify, or remove event channels as shown in the following examples.

**Creating a new event channel**

The following example creates a channel to receive events whenever a process instance or workstep instance is suspended from the specified process templates.

```java
EventChannel evtchn = new EventChannel("jms/MyQueue", "queue");
evtchn.addFilterCondition(EventChannel.EVENT_VALUE, "PI_SUSPENDED, W_SUSPENDED");
evtchn.addFilterCondition(EventChannel.PT_NAMES, "Hiring, Approval, Assignment");
eventChannel.create();
```

**Modifying an event channel**

The following example shows how the above channel is modified.

```java
EventChannel evtchn = EventChannel.get("jms/MyQueue");
evtchn.setChannelExpirationDate( (new Date()).getTime() + 86400000); // today+1day
evtchn.save();
```

**Removing an event channel**

The following example shows how you can remove a channel.

```java
EventChannel evtchn = EventChannel.get("jms/MyQueue");
evtchn.remove();
```
Event filters

The EventPublisher provides ANDFilter as the default filter. You can override this option for all event channels by specifying a new Filter class. You can also specify a different filter class for each channel. Note that all filter classes should implement the Filter interface.

The ANDFilter requires that all filter conditions specified for an event channel must be true. If any one of the filter conditions is evaluated to false, then the event does not publish to that JMS destination.

The default ANDFilter class is specified in the property file as shown below:

```
publisher.default.filter.class= com.savvion.sbm.eventpublisher.filter.ANDFilter
```

When filter conditions are not specified for an event channel, the Event Publisher invokes the default filter class to validate an input event. When filter conditions and filter class name are specified for an event channel, the Event Publisher uses only the specified filter class for the channel.

Prefilter conditions

The Event Publisher allows prefilter conditions on the following event data.

- Event type
- Event value
- Process template names
- Workstep names

You can specify multivalue data for each property. For example, you can specify comma separated event values for an event channel.

- Example1: Filter conditions for channel C1.

```
EVENT_TYPE: BP Server
EVENT_VALUE: P_INSTALLED, P_REMOVED, P_SUSPENDED
PT_NAMES: Hiring, Approval, Assignment
```

When one of the specified events is generated for the specified process template, then that event is published to the specified channel C1. The same events generated for any other process template are not sent to this event channel.

- Example2: Filter conditions for channel C2.

```
EVENT_TYPE: BP Server
EVENT_VALUE: W_ACTIVATED, I_COMPLETED, W_COMPLETED
WS_NAMES: Feedback
```

When a workstep called Feedback is activated or completed, that event is sent to channel C2. Note that any process template with a workstep called Feedback satisfies this condition. In other words, all work items completed for a workstep named Feedback, from any process template, are eligible for sending the event to channel C2.
Prefilter conditions on event context data

The default ANDFilter also supports filter conditions on any data from event context. All filter conditions using event context should be specified as a name value pair in a Java Map object.

Methods supported

The value object EventChannel supports the following methods to specify the filter conditions on primary data, or using event context data.

```java
EventChannel chn = new EventChannel("sbmTopic", EventChannel.TOPIC);
Chn.addFilterCondition(EventChannel.EVENT_TYPE, "BP Server,BPM Events");
Chn.addFilterCondition(EventChannel.EVENT_VALUES, "P_CREATED,P_INSTALLED, P_REMOVED");
Chn.addFilterCondition(EventChannel.PT_NAMES, "Hiring,Approval,Assignment");
Chn.addFilterCondition(EventChannel.WS_NAMES, "worstep1,workstep2,workstep3");
Chn.addFilterCondition(EventChannel.EVENT_CONTEXT, Map nameValuePair);
chn.removeFilterCondition(EventChannel.EVENT_TYPE);
chn.removeFilterCondition(EventChannel.EVENT_VALUES);
chn.removeFilterCondition(EventChannel.PT_NAMES);
```

If a context filter is to be set for EventChannel.EVENT_CONTEXT key, then its value should be Map. For example, in the above code, consider

```java
Chn.addFilterCondition(EventChannel.EVENT_CONTEXT, Map nameValuePair);
```

A possible nameValuePair would be contextFilterMap with the following additional code:

```java
HashMap contextFilterMap = new HashMap();
Vector PI_Values = new Vector();
PI_Values.add(new Long("1428"));
contextFilterMap.put("CTX.PROCESSINSTANCEID", PI_Values);
eventChannel.addFilterCondition(EventChannel.EVENT_CONTEXT, contextFilterMap);
```

The HashMap needs a key of type `CTX.PROCESSINSTANCEID` where the value should be Vector.

Post-filter conditions

JMS supports message selectors to allow a JMS consumer more selectivity about the message it receives from a particular destination. Message selectors are based on a subset of the SQL92 conditional expression syntax. Message selectors use message properties and headers as criteria in conditional expressions. JMS clients can use message properties to filter messages.

The Event Publisher adds the following properties to each published JMS message.

- **EVENT_TYPE**
- **EVENT_VALUE**
- **PROCESSTEMPLET**
- **PROCESSINSTANCEID**
- **WORKSTEPNAME**
- **CREATOR**
The above values are keys of the property value added to each JMS message. A JMS client may use the following query to filter messages from a destination.

```java
String selector = "PROCESSTEMPLATE in ('Hiring') AND CREATOR = 'ebms' ";
TopicSubscriber subscriber = Session.createSubscriber(sbmTopic, selector, false);
```

## Event publisher example

This section provides an example to write a client code to subscribe events from the already created Event Channels.

You can refer to the example code for details at OEBPS_HOME\bpserver\examples\EventPublisherExample.zip.

```java
private static void subscribeBPServerMessages()
{
    Session tsession = null;
    try {
        BMJMSService jmsService = new BMJMSService(factory_name, provider);
        //post-filter expression based on SQL92
        String postFilter = "";
        if((appName != null) && (!appName.equals(""))) {
            postFilter = "PROCESSTEMPLATENAME = " + appName + "";
        }
        // first parameter is connection factory (if null take default)
        // second and third parameter is userName and password. By default security is not enabled.
        javax.jms.Connection conn = jmsService.createConnection(null, user, password);
        tsession = jmsService.createSession(conn);
        //Example for Post Filter on event:::
        //postFilter = "PROCESSTEMPLATENAME = 'Hiring' AND EVENT_VALUE = 'I_COMPLETED'";
        //Create the subscriber with the above message filter
        javax.jms.MessageConsumer tsubscriber = jmsService.createMessageConsumer(tsession, eventPublisherTopic, postFilter);
        while (true) {
            /* Subscribe the message from the topic in a loop*/
            Message msg = tsubscriber.receive();
            if(msg.propertyExists("PROCESSTEMPLATENAME")) {
                System.out.println();
                System.out.println("Message received");
                System.out.print("ProcessTemplateName=");
                System.out.println(msg.getStringProperty("PROCESSTEMPLATENAME"));
                if(msg.propertyExists("PROCESSINSTANCEID")) {
                    System.out.print(" | ProcessInstanceID=");
                    System.out.println(msg.getLongProperty("PROCESSINSTANCEID"));
                }
                if(msg.propertyExists("WORKSTEPNAME")) {
                    System.out.print(" | WorkStepName=");
                    System.out.println(msg.getStringProperty("WORKSTEPNAME"));
                }
                catch (JMSException e) {
                    System.out.println("Exception occurred: " + e.toString());
                }
            }
            finally {
            try {
                //close the session properly to avoid memory leak
                if (tsession != null) {
                    tsession.close();
                }
            } catch (JMSException e) {
                System.out.println("Exception occurred while closing Topic Session: " + e.toString());
            }
        }
    } catch (JMSException e) {
        System.out.println("Exception occurred: " + e.toString());
    }
```
Administering the event publisher

You can start/stop the EventPublisher module, change its properties, and manage its log file.

Starting/Stopping event publisher

You must have the JMS server up and running to start and stop the EventPublisher.

• On Windows, go to OEBPS_HOME\bin.
  To start the EventPublisher, execute `startEventPublisher [-u <user>] [-p <passwd>]`
  To stop the EventPublisher, execute `stopEventPublisher [-u <user>] [-p <passwd>]`

  **Note:** In case user and password values are not provided, the default values “ebms” and “ebms” are used.

• On UNIX, execute `eventpublisher.sh start` to start the EventPublisher and `eventpublisher.sh stop` to stop the EventPublisher.
  You can also start or stop the EventPublisher by invoking the following static methods.

  • `EventPublisherClient.start();`
  • `EventPublisherClient.stop();`

Modifying event publisher properties

You can modify the following properties at runtime.

```
publisher.default.filter.class=
publisher.event.cmdch.size=
publisher.event.transaction.size=
publisher.sleep.time=
publisher.debug.engine=
publisher.printEvent=
publisher.customclass.location=
```

After modification, you must inform the EventPublisher module that some properties were changed and cached values should be refreshed. Execute the following command to refresh the properties.

• Go to OEBPS_HOME\bin and execute `eventpublisher.cmd refresh`.

You can also refresh the properties of the EventPublisher by invoking the following static method.

• `EventPublisherClient.refresh();`

  **Note:** Changes to any other property is effective only during the subsequent startup of the EventPublisher module.
Managing the log file

The EventPublisher uses the Apache Log4j library for logging debug messages. All debug and exception messages are written to `eventpublisher.log` in `OEBPS_HOME\logs`. You can print the debug messages to the log file by setting the log level to DEBUG, and also by setting the following parameters in `OEBPS_HOME\conf\bpserver.conf` as follows:

```
publisher.debug.engine=true
publisher.printEvent=true
```

Error handling

When a database error occurs because of invalid connections, the EventReader continuously polls the database until the connection is restored. This handling is especially required during shutdown initiation of the Event Publisher while it is polling the database.

EventReader stops the Event Publisher for any other database errors.

When EventReader stops due to unrecoverable database errors, we recommend you look at the log file for detail error messages, and correct them manually. For example, the error may occur because of an invalid JMS configuration or restricted access permission to JMS destinations. In these cases you must modify the required configuration.
This chapter explains the mechanism of the messaging workstep and the features supported by this workstep. A messaging workstep is broadly divided into two types: message publisher workstep and message subscriber workstep. A message publisher workstep is a workstep that publishes and broadcasts a predefined message when the triggering event takes place. A message subscriber workstep is a workstep that subscribes to, and listens on a particular channel, and performs a predetermined action on receipt of a predefined message.

For details, see the following topics:

- Message
- Message descriptor
- Creating MessageDescriptor
- Channel
- Message publisher workstep
- Message subscriber workstep
- Process instance creation on a message
- Messaging example
Message

Message is the vital link between the message publisher workstep and the message subscriber workstep. A message is a self-contained and self-explanatory package of business data which provides the necessary data and context information to its recipient to carry out its task independently. Typically, a message has two parts — header and payload.

The header contains name-value pairs of data pertaining to the sender and receiver of the message.

The payload of a message contains the actual business data, for example, purchase order data in XML format. Business Process Server supports only XML data for payload.

JMS based message

Currently Business Process Server supports only JMS based messages. In JMS based messages, the following data types are supported:

- CHARACTER
- INTEGER
- INT64
- LOGICAL
- DECIMAL

Message descriptor

A message descriptor provides the description of a message in XML format. The messages with different XML Schema have unique names. In addition to the unique message name, the message descriptor also contains, the message template and its namespaces, meta-data for the message header and the message payload, and message correlation properties. All the properties defined in the `<Header>` and `<Payload>` sections of a message descriptor should have unique names.

A broad structure of a Message Descriptor is shown below.

```
<MessageDescriptor name="{name to identify a message}">
  <Template location="" encoding="base64"/>
  <Header/>
  <Payload>
    <Property name="{prop-name}" type="java.lang.String">
      <XPath></XPath>
    </Property>
  </Payload>
  <Correlations>
    <Correlation name="{corr-name}"/>
  </Correlations>
  <Namespaces/>
</MessageDescriptor>
```

Each section of the message descriptor is described in the following sections. An example of a message descriptor is given in the following section.
Example of a message descriptor

```xml
<MessageDescriptor xmlns:xs="http://www.w3.org/2001/XMLSchema-instance"
xs:noNamespaceSchemaLocation="MessageDescriptor.xsd">
  <Message name="NewProduct">
    <Template>
      <Product>
        <Id/>
        <Name/>
        <Price/>
      </Product>
    </Template>
    <Header>
      <Property name="ReceiverAppName"/>  
      <Property name="ReceiverInstanceId" type="long"/>
      <Property name="ReceiverTaskName"/>
      <Property name="ReceiverPriority"/>
      <Property name="ReceiverInstanceAlias"/>
      <Property name="SenderAppName"/>
      <Property name="SenderTemplateName"/>
      <Property name="SenderInstanceId" type="long"/>
      <Property name="SenderTaskName"/>
      <Property name="SenderEmail"/>
      <Property name="MySenderProp" type="java.lang.String"/>
    </Header>
    <Payload>
      <Property name="product" type="java.lang.String"> //to get the entire payload as XML
        <XPath>/Product</XPath>
      </Property>
      <Property name="productDetail" type="java.lang.String"> //to get part of the payload as XML
        <XPath>/Product/Name</XPath>
      </Property>
      <Property name="prodId"> //to read a specific value in payload
        <XPath>/Product/Id/text()</XPath>
      </Property>
      <Property name="prodName">
        <XPath>/Product/Name/text()</XPath>
      </Property>
      <Property name="price" type="long">
        <XPath>/Product/Price/text()</XPath>
      </Property>
    </Payload>
    <Correlations>
      <Correlation name="ProductPK">
        <Property name="prodId"/>
      </Correlation>
    </Correlations>
  </Message>
</MessageDescriptor>
```

**XML template**

The `Template` tag within `MessageDescriptor` may contain an inline XML template of the payload. Typically, a template is a valid XML file with blank tags, that is, tags which contain no data. Wherever values are provided for XML tags, these are used as the default values at runtime. While defining a message template, the user can provide either the entire XML template inlined within the descriptor or an HTTP URL to the template. Two examples of 'Template' tag are shown below.

- A template example with a HTTP URL.
• A template example with a inline content.

```xml
<Template encoding="base64">
    <Product>
        <Id/>
        <Name/>
        <Price/>
    </Product>
</Template>
```

The encoding attribute is required only for the inline template. Currently, BP Server supports the Base64 encoding only.

**XML instance**

When you enter values for various XML elements in an XML Template, it becomes an XML Instance. An XML Instance is the actual XML file containing all the data needed to send it as a JMS message. In a publisher workstep, the values for these XML tags of a template are copied from the specified dataslots in the workstep.

**Header**

The properties specified within the `Header` tag constitute the header of the JMS message. The header properties can include predefined system properties and user defined custom properties. In addition to the list of system properties provided by BP Server, user can define any custom property in header.

```xml
<Header>
    <Property name="approverName"/>
</Header>
```

The list of predefined system properties is given below:

• MessageName
• ReceiverAppName
• ReceiverInstanceId
• ReceiverTaskName
• ReceiverPriority
• ReceiverInstanceAlias
• SenderAppName
• SenderTemplateName
• SenderInstanceId
• SenderTaskName
• SenderEmail
Note: You cannot modify the name and type of the system properties.

Example:

```
<Header>
  <Property name="MessageName"/>
  <Property name="SenderEmail"/>
  <Property name="approverName" type="java.lang.String"/> //custom property
</Header>
```

The default data-type for a property is assumed as `java.lang.String` type. If the property data-type is other than `java.lang.String`, then the `type` attribute should be specified with a proper value.

System properties

The system properties are the system defined properties with specific names and types, which cannot be modified. The details of different system properties are explained in Table 20: System properties on page 199.

Table 20: System properties

<table>
<thead>
<tr>
<th>System property</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MessageName</td>
<td>java.lang.String</td>
<td>When the message is published by BP Server, it always adds these system properties to the message header and therefore, no dataslot mapping is required.</td>
</tr>
<tr>
<td>SenderAppName</td>
<td>java.lang.String</td>
<td>This system property is optional, and the user needs to provide dataslot mapping for this property. The e-mail address is used to notify about issues and also when the message is discarded by the system.</td>
</tr>
<tr>
<td>SenderTemplateName</td>
<td>java.lang.String</td>
<td>These system properties are optional, and the user needs to provide dataslot mapping for them. User can define these properties if the target is known and therefore, these can be used for finding the target of the published message.</td>
</tr>
<tr>
<td>SenderInstanceId</td>
<td>long</td>
<td></td>
</tr>
<tr>
<td>SenderTaskName</td>
<td>java.lang.String</td>
<td></td>
</tr>
<tr>
<td>SenderEmail</td>
<td>java.lang.String</td>
<td></td>
</tr>
<tr>
<td>ReceiverAppName</td>
<td>java.lang.String</td>
<td>These system properties are optional, and the user needs to provide dataslot mapping for them. These properties define the alias and priority of the instance to be created using a message.</td>
</tr>
<tr>
<td>ReceiverInstanceId</td>
<td>long</td>
<td></td>
</tr>
<tr>
<td>ReceiverTaskName</td>
<td>java.lang.String</td>
<td></td>
</tr>
<tr>
<td>ReceiverInstanceAlias</td>
<td>java.lang.String</td>
<td></td>
</tr>
<tr>
<td>ReceiverPriority</td>
<td>java.lang.String</td>
<td></td>
</tr>
</tbody>
</table>

Custom properties

User can provide the custom header properties (for example, approverName). The user needs to specify the values for these properties by providing proper dataslot mappings.
Payload

The **Payload** tag contains one or more payload properties. Each property should have a unique name and an XPath pointing to a valid location in the payload XML.

```xml
<Payload>
  <Property name="prodName">
    <Xpath>/Product/Name/text()</Xpath>
  </Property>
  <Property name="price" type="double">
    <Xpath>/Product/Price/text()</Xpath>
  </Property>
</Payload>
```

In the above example, the property `prodName` is an alias to the value of the XPath `/Product/Name/text()`, that is, it is the alias for the value of XML tag `<Name>` inside the tag `<Product>` in the message XML.

The default data-type for a property is `java.lang.String`. If a property's data-type is other than `java.lang.String`, then the ‘type’ attribute should have a proper java type value.

The mapping of various payload properties to the XML tag and data in an example XML message is shown in **Figure 24: NewProduct.XML file** on page 200.

**Figure 24: NewProduct.XML file**

```
NewProduct.xml
```

```
<Namespace alias="po" uri="http://www.progress.com/PurchaseOrder"/>
<Namespace alias="prd" uri="http://www.progress.com/Product"/>
```

**Namespace**

If the XPath query for a payload property uses a namespace alias, then the same alias should also be defined in ‘Namespaces’ tag as shown in the following example.

```xml
<Namespaces>
  <Namespace alias="po" uri="http://www.progress.com/PurchaseOrder"/>
  <Namespace alias="prd" uri="http://www.progress.com/Product"/>
</Namespaces>
```

The Message Manager automatically extracts all the namespaces used in the XML and save them in the message descriptor.
Creating MessageDescriptor

MessageDescriptor can be created using the Message Manager from the Progress Developer Studio for OpenEdge. In the Progress Developer Studio for OpenEdge, click Project and then Message Manager. This opens the Message Manager dialog box. Click the New button to open the Add New Message dialog box. This dialog box has text fields for the Message Name and Description.

It has separate tabs for each of the five sections of the MessageDescriptor. For more information about MessageDescriptor, see Message descriptor on page 196.

Template tab

User can type in the XML template content in the text area. Optionally, use the Browse button or type in the http URL location in the location text field to load the template content from a file in the local file system.

The Load button reads the content from the specified location and displays it in the text area. If the location is an http URL, then the content shown in the text area is non-editable. If the content is loaded from local file system or a file URL, then it can be edited. An example of Add New Message dialog box with Template tab is shown in Figure 25: Add new message window -- Template tab on page 201.

Figure 25: Add new message window -- Template tab

![Template tab example](image)

Header tab

User can add Header properties from the Message Descriptor in this tab. The names of the properties should be unique across header properties and payload properties.
The header properties can be classified mainly as the system properties and custom properties. For more information about the list of predefined properties used by BP Server, see System properties on page 199. An example of Add New Message dialog box with Header tab is shown in Figure 26: Add new message window -- Header tab on page 202.

Figure 26: Add new message window -- Header tab

![Image of Add New Message dialog box with Header tab]

Payload tab

Payload tab has two sections. The upper section displays payload properties in a tabular structure and the lower section displays the XML template content in a tree format. An example of Add New Message dialog box with Payload tab is shown in Figure 27: Add New Message window -- Payload tab on page 202.

Figure 27: Add New Message window -- Payload tab

![Image of Add New Message dialog box with Payload tab]

User can add a new payload property by right clicking on any node in the tree. This opens the dialog box shown in Figure 28: Add payload properties on page 202. User can add a new payload property using this dialog box.

Figure 28: Add payload properties

![Image of Add Payload Property dialog box]
For advanced options, user can access the XPath wizard (as shown in Figure 29: Add payload properties -- Advanced on page 203) by clicking the Add button. The XPath wizard has advanced features like evaluating the XPath before adding the properties.

**Figure 29: Add payload properties -- Advanced**

![Add Payload Property](image)

**Correlations tab**

Correlations tab is used to add correlations for the Message Descriptor. Correlations are used only for the Message Subscriber workstep. An example of Add New Message dialog box with Correlations tab is shown in Figure 30: Add new message window -- Correlation tab on page 203.

**Figure 30: Add new message window -- Correlation tab**

![Add New Message](image)
Namespaces tab

This tab displays all the namespaces defined within the template XML. These namespaces are also stored as a part of message descriptor. An example of Add New Message dialog box with Namespace tab is shown in Figure 31: Add New Message window -- Namespace tab on page 204.

Export and import of message descriptor

When a Message Descriptor is created in the Progress Developer Studio for OpenEdge, it is saved only on the computer where Progress Developer Studio for OpenEdge is running. To use this message descriptor in a process template installed on a Business Process Server (either in a publisher or a subscriber workstep), it should be exported to the Business Process Server. Similarly, if there is a need to view or modify a Message Descriptor stored on Business Process Server, then it can be imported in the Progress Developer Studio for OpenEdge. These operations are performed by using the Message Manager dialog box of the Progress Developer Studio for OpenEdge. A sample of this dialog box is shown in Figure 32: Message manager window on page 204.

The ‘Local’ panel is meant for all the operations that are effective only on the local computer where Progress Developer Studio for OpenEdge is running. All the message descriptors are saved as XML files on local computer in the <WRK_DIR>/com.savvion.studio/conf/resources/messaging/messages folder.
The usage of the buttons in the Local panel is explained below.

- **New** — Creates a new Message Descriptor and saves it on the local computer.
- **Edit** — Modifies a Message descriptor from the local computer.
- **Delete** — Deletes a Message Descriptor from the local computer. This does not delete it from the Business Process Server Storage.

The 'Remote' panel is meant for all the operations that are effective only on the remote computer where Business Process Servers are running. The Portal server must be started before performing these operations. The usage of the buttons in the Remote panel is explained below.

- **Export** — Exports the Message Descriptor from local computer to Business Process Server Storage for persistence.
- **Import** — Imports the Message Descriptor from Business Process Server Storage to local computer.
- **Delete** — Deletes the Message Descriptor from Business Process Server Storage.

## Channel

BP Server requires the following information to publish a message to the specified destination:

- JMS Topic Name or Queue Name
- Server Properties
- Factory class
- Provider URL
- Principal
- Credential

The provider URL includes variable information, like host-name, port number. In general, any change in the server properties should not force you to make any change in the message publisher workstep definition. This means, the above properties required for a message destination cannot be stored in the publisher workstep. Instead, it uses alias for all these properties with a unique channel name as described below:

```xml
<ChannelDescriptor>
  Channel type="JMS" name="{channel-name}">
    Description/>
    <Destination type="QUEUE">{dest-name}</Destination>
    <Server local="false">
      <FactoryClass/>
      <Provider/>
      <Principal/>
      <Credential/>
    </Server>
  </Channel>
</ChannelDescriptor>
```

As shown above, a channel is an alias that identifies a JMS destination and its server properties, where the JMS destination is accessible. The `<Server>` tag is optional for the local server, where BP Server is running. When no server properties are defined for a channel, BP Server automatically looks up the destination within the local server.
An example:

```xml
<ChannelDescriptor>
  <Channel type="JMS" name="ProductChannel">
    <Destination type="QUEUE">jms/ProductQueue Destination</Destination>
    <Server local="true"/>
  </Channel>
</ChannelDescriptor>
```

**BP Server channel**

A default channel "BP ServerChannel" is available in the system to be used by the publisher workstep. The destination of the default channel is same as the default queue used for message subscriber workstep. User should use this channel to publish messages to subscriber worksteps in other business processes.

**Note:** Communication between two BP Server processes should be only through a JMS Queue.

**Message publisher workstep**

*Figure 33: Message Publisher Workstep* on page 206 shows how inputs to the Message Publisher Workstep dialog box are used for the XML.

**Message Selection:** In a publisher workstep, user has to first select a single named message to be published from the provided list of named messages. For the selected message, a dataslot for message properties (payload as well as header) to be mapped should also be selected. BP Server automatically displays the list of message properties in a drop-down list beside the dataslot option.

**Channel Selection:** Once the dataslots and message properties mappings are defined, then user may choose a channel name to which the message should be published upon activation of the workstep.

The message publisher workstep performs the following operations on activation:

1. Construct the JMS message from the XML message template.
2. Add the JMS message header from the header properties.
3. Get the channel/destination to which the message is to be sent.
4. Publish the message on this channel.

XML tags for publisher workstep

An example of how the XML tags are to be used for a publisher workstep is shown below.

```xml
<MessagePublisherWS name="PublishNewProductMessage">
  <Duration value="7200">2h</Duration>
  <Priority>medium</Priority>
  ...
  <MessageChannel>BP ServerChannel </MessageChannel>
  <Messages>
    <Message name="NewProduct">
      <MessageData>
        <Copy to="SenderComments">Sending Messages For Testing</Copy>
        <Copy to="prodId">@ds1</Copy>
        <Copy to="price">@ds3</Copy>
      </MessageData>
    </Message>
  </Messages>
</MessagePublisherWS>
```

Construction of JMS message

The JMS message constructed by the publisher workstep is composed of a JMS message payload and a JMS header.

Creation of JMS message payload

First, the publisher creates an instance of the XML Template by populating the template with dataslot values. It identifies the XML elements or locations to be populated with data in the XML Template, by the message properties specified for this workstep. These message properties contain the XPath location of the XML element as defined in the message metadata file. The message properties specified in the workstep are mapped to dataslots or constants, so that the XPath for the message property is replaced by the corresponding dataslot. The XML instance thus created forms the payload of the JMS message. If an XPath element is not mapped and a default value is defined for the element in the template, then this default value is used.

Creation of JMS message header

The publisher workstep adds the JMS header property MessageName and assigns it the value of message name. Also, the following system properties are automatically added to identify the sender.

- SenderAppName
- SenderTemplateName
- SenderInstanceId
- SenderTaskName

The receiver can use these properties to communicate with the sender on need basis. For example, the receiver can send a reply message to the sender using the above header information.
Apart from the above system properties, user can optionally add any custom property (available in the message metadata file) to the message header for dynamic content and map it to the datalslots or constants.

**Publisher message handler**

For a Message Publisher workstep, user may optionally provide a Message Handler class. If no Message Handler class is provided by the user, then BP Server uses the following default Publisher Handler class: `com.savvion.sbm.bizlogic.messaging.publisher.BLJMSMessagePublisher`

You can implement a Custom Message Handler by using one of the following two alternatives:

- The custom Message Handler class can extend abstract class `com.savvion.sbm.bizlogic.messaging.publisher.MessagePublisher`
  
  The custom handler class then has to provide implementation for the following abstract methods:
  
  - public abstract void init();
  - public abstract void destroy();
  - public abstract Object createMessage() throws Exception;
  - public abstract HashMap publish(Object message) throws Exception;

- The custom Message Handler class can alternatively extend class `com.savvion.sbm.bizlogic.messaging.publisher.BLJMSMessagePublisher`

  The `BLJMSMessagePublisher` class already implements all the abstract methods described in the previous alternative explained above. The custom Handler class can either make use of these methods or can customize the behavior by overriding them.

The following code snippet shows a custom Handler class extending from `BLJMSMessagePublisher`. Please note that in the `publish()` method, first this we call is `super.publish()`, thus publishing the JMS Message and then some debug information is printed.

```java
public class MyPublisherHandler extends BLJMSMessagePublisher {
    public HashMap publish(Object message) throws Exception {
        HashMap map = super.publish(message);
        System.out.println("Process Instance Name: " + getProcessInstanceName());
        System.out.println("Process Instance ID: " + getProcessInstanceID());
        System.out.println("WorkStep Name: " + getWorkStepName());
        return map;
    }
}
```

When a custom Handler class is provided for a Publisher workstep, BP Server does not do anything except instantiating an object of the Handler class and invoking its methods. The responsibility of constructing the message and publishing it, lies with the Handler class. Alternatively, as shown in the above code snippet, a call to the method `super.publish()` publishes the JMS message, and then any additional custom logic can be added.

**Error condition for publisher message handler**

When an Error or an Exception is encountered while publishing a message by a Message Publisher Workstep Instance, the workstep is suspended and the message is not published. When the error condition is resolved and then, if the publisher workstep is resumed, then the message gets published.
BP Server operations on activation of a publisher workstep

BP Server performs the following operations when a publisher workstep is activated.

1. Create a JMS message with header containing message name as the value of the header property MessageName. BP Server also adds other system properties to identify the sender. For example, SenderAppName, SenderTemplateName, SenderInstanceId, SenderTaskName.

2. Add other properties in the JMS header as specified in the publisher workstep.

3. Create an XML instance of the specified XML Template using the payload properties chosen from the metadata in the publisher workstep. The value of the property (which points to an XPath expression in the metadata) is taken from the corresponding mapped dataslot.

4. Add the XML instance as the payload of the message created in step 1.

5. Publish the fully constructed message to the specified channel (message destination).

Message subscriber workstep

The message subscriber workstep in process flow is activated only when the subscribed message is received; otherwise it remains in the WAIT state. In this workstep, for the subscribed message, mapping from message properties (both header as well as payload) to dataslots should be defined for automatic data assignment. When the requisite message is received, BP Server extracts the data from the message using the mapping and XPath information into the dataslots. Then, it invokes the associated handler (if any,) to process the received message.

Message properties and dataslot mappings

For the subscribed message, the message properties in the payload of the message meta-data may be mapped to the dataslots of the process. BP Server uses this mapping information to extract the data from message payload using the XPath specified in the message file, and copy the extracted data to the corresponding mapped dataslot.

Similarly, the header properties in the message meta-data can also be mapped to the dataslots of the process. This mapping information is used to fetch the value of the header property and copy it to the corresponding mapped dataslot.

The received message is processed for data extraction as per the order specified in the message subscription list. BP Server invokes the message handler with the received message as input to the handler only after extracting data from the message and copying to the specified dataslot. The message handler for the "Message Subscriber WorkStep" is optional. Since BP Server already takes care of the data extraction from the message based on the mapping definitions, the handler is required only when any additional processing of the received message is needed.

XML tags for subscriber workstep

An example of how the XML tags are to be used for a subscriber workstep is shown below.

```xml
<MessageSubscriberWS name="MsgWaitWS">
  <Duration value="7200">2h</Duration>
  <Priority>medium</Priority>
  <OnMessage waitFor="ANY">
    <MessageHandler>com.savvion.test.MessageProcessor</MessageHandler>
  </OnMessage>
  <Messages>
```

In the above example, the first `<Copy>` tag copies the value of the JMS Header property `SenderInstanceId` into the dataslot `SenderID`. The remaining `<Copy>` tags copy the data from the message properties `prodId`, `prodName` and price to the dataslots `ds1`, `ds2` and `ds3` respectively. The value for the message property "prodId" is retrieved from the message payload using the specified XPath expression in the message file.

**Message correlation**

When the BP Server receives a message, it requires a mechanism to identify the workstep instances to which this message is to be delivered. The rules which help to identify the correct instances to deliver the received message are called correlation. The message correlation is based on comparison of data from the message with dataslot values.

**Named correlations**

A named correlation defines a set of message properties that are used to correlate the received message with the target workstep instances.

**Message definition**

The message definition is extended to include one or more named correlations as described below:

```xml
<MessageDescriptor name="NewProduct">
  <Header/>
  <Payload>
    <Property name="prodId">
      <XPath>/Product/Id/text()</XPath>
    </Property>
    <Property name="prodName">
      <XPath>/Product/Name/text()</XPath>
    </Property>
  </Payload>
  <Correlations>
    <Correlation name="productId">
      <Property name="prodId"/>
    </Correlation>
  </Correlations>
</MessageDescriptor>
```

**Correlation definition**

The above product example is extended below to define a named correlation using `prodId`.

```xml
<MessageDescriptor name="NewProduct">
  <Header/>
  <Payload>
    <Property name="prodId">
      <XPath>/Product/Id/text()</XPath>
    </Property>
    <Property name="prodName">
      <XPath>/Product/Name/text()</XPath>
    </Property>
  </Payload>
  <Correlations>
    <Correlation name="productId">
      <Property name="prodId"/>
    </Correlation>
  </Correlations>
</MessageDescriptor>
```
Correlation using message payload and process dataslots

The named correlation "ProductPK" defined for the "NewProduct" message is used in the subscriber workstep. A named correlation may have more than one key. A message descriptor may contain one or more named correlations. However, a workstep should use at most only one named correlation. Each correlation key should be mapped to process dataslot that provides the value for the message correlation at runtime.

When a named message is received for a workstep instance, the values of all correlation keys are compared with the value of the mapped dataslot and the message is delivered to the workstep instance only when all comparisons are successful.

When no correlation is defined for a subscribed message, the message is delivered to the target workstep instance unconditionally, provided the name of the received message is same as the subscribed message of the workstep. This means, correlation is optional for both; the messages as well as for the subscriber workstep.

Restrictions for message correlation

• Message correlation can be done only using Message Payload properties. The Message Header properties cannot be used for Message correlation purpose.

• Constant value cannot be provided for a Correlation property in a Message Subscriber workstep. A Correlation property can be mapped only to a dataslot for a Message Subscriber workstep.

• Message Correlation functionality cannot be applied for Process Instance creation on receiving a message on BP ServerChannel. For more information, see Process instance creation on a message on page 214.

Subscriber message handler

It is optional to provide a Message Handler class for a Subscriber workstep. If a custom Handler class name is mentioned in the Subscriber workstep properties, then the custom Handler class should implement interface com.savvion.sbm.bizlogic.messaging.MessageHandler

The custom handler class has to provide implementation for following methods from interface:

• public void init();
• public void setContext(ProcessContext context);
• public HashMap execute(List bizMessage);
• public void destroy();
• public HashMap getMessagesToDiscard();
When a custom handler class is provided for a Message Subscriber workstep, BP Server engine first delivers the message to the appropriate waiting subscriber workstep by performing filtering and correlation, and it also updates the dataslot values using the message payload properties. Then BP Server engine instantiates an object of the Handler class and invokes Handler class methods.

Sometimes, the message may not be in the expected format or there can be some error in the message structure or data. In such cases, the message must be discarded without completing the workstep. To do this, user can implement getMessagesToDiscard() method in the custom handler class. This method returns a list of messages to be discarded in a HashMap. After invoking other Handler methods, BP Server checks if any messages are to be discarded by calling this method. If the HashMap returned by this method call is non-empty, then BP Server discards those messages and keeps the workstep in a wait state. A code snippet of a Handler class, where all messages having 'Price' property value greater than 5000 are discarded, is shown below.

```java
public class MyHandler implements MessageHandler {
    private BizMessage message = null;
    private HashMap messageToDiscard = new HashMap();
    public void init() {
    }
    public void setContext (ProcessContext context) {
        System.out.println("getInputSlots: "+ context.getInputSlots());
    }
    public HashMap execute (List bizMessage) {
        for (Object obj : bizMessage) {
            BizMessage message = (BizMessage) obj;
            if (message.getDoubleProperty("Price") > 5000.0) {
                messagesToDiscard.put(message, "Price too high!");
            } else {
                // process the message
            }
        }
        return new HashMap();
    }
    public void destroy() {
    }
    public HashMap getMessagesToDiscard() {
        return messagesToDiscard;
    }
}
```

**Target process information**

Whenever a Business Process Server application receives a message, BP Server performs the following operations:

1. Find all process templates that have subscribed to this message in the start workstep and create instances for those process templates.
2. Search all process templates to find the list of all worksteps that are subscribed to the received message.
3. Find all existing process instances of such process templates. The process instances can be in a suspended or activated states.
4. Find all qualified subscriber workstep instances of these process instances found in step 3 by applying correlation condition, if any. BP Server ensures delivery of the message to these qualified workstep instances.
5. Save a reference of the message for each qualified workstep instance found in step 4.

BP Server ensures to deliver the message only to those Process Instances, which are in active or suspended state. It cannot deliver the message for process instances that are created in future.
In the above operations, the first one requires BP Server to search all process templates of the system. This search can be narrowed down if the sender of the message specifies any of the following process data in the JMS header. These are called system properties that can be used by BP Server engine to narrow down the search for the qualified targets.

- ReceiverInstanceId, and ReceiverTaskName
- ReceiverAppName, and ReceiverTaskName
- ReceiverAppName

In a publisher workstep, user can add these properties to the message header. For more information about this, see Message publisher workstep on page 206. When a message with the receiver information is received in a subscriber workstep, BP Server engine uses this data to narrow down the search to find out the qualified targets of this message.

**Error condition for target process information**

When an Error or an Exception is encountered while receiving a message by a Message Subscriber Workstep Instance, the workstep is suspended. When the error condition is resolved and then, if the subscriber workstep is resumed, then the same message is used to complete the subscriber workstep instance.

**Sequence of steps for message subscriber worksteps**

BP Server performs the following operations for the message subscriber workstep:

1. Mark the subscriber workstep instance with WAIT state if the required message was not received at the activation time of the workstep.
2. Wait until the required message is received and stored in the database.
3. When the message is received, mark the workstep instance with ACTIVATED state.
4. From the message, extract the data from the payload using the XPath and copy it to the mapped dataslot specified in the message-dataslot mappings. Also extract the data from the message header and copy it to the corresponding mapped dataslot.
5. Invoke the message handler with the received message.
6. Complete the workstep instance and activate the next workstep instance of the process flow.

BP Server automatically discards a message in any of the following conditions:

- When there are no active message subscriber worksteps subscribing to such a message.
- When there are no active process instances that can activate a subscriber workstep in future.

**What if the received message is invalid?**

Consider the scenario that the message received by BP Server is invalid, for example,

- the message payload does not contain the required data at all or the format of the data is incorrect.
- the handler finds that the message is invalid.

In such cases, if the received message contains the sender e-mail address under the JMS header property `SenderEmail`, then the sender is notified about the invalid message by e-mail. The sender can construct a valid message using this information and resend the message.
Workstep State

The Message Subscriber Workstep (MSW) is in a WAIT state until the subscribed message is received. The MSW goes into activated state as soon as the message is received because it is not waiting for the messages as a condition to activation. Hence, in MSW, when the messages are consumed and the handler is invoked, the workstep is in activated state.

Process instance creation on a message

BP Server supports instance creation using the incoming messages on BP ServerChannel. For every message received by BP ServerChannel, a new process instance can be created for those process templates, whose start workstep has subscribed to the same message type.

Start workstep message subscription

A start workstep of a template can subscribe to only one message. The creation of a process instance, optionally requires the priority and the process instance name prefix. If the JMS message header carries the property ReceiverPriority, then its value is used as the priority of the new process instance. Similarly, ReceiverInstanceAlias is used as the process instance name alias. If these properties are not present, then the default values are used.

The dataslot values of the new process instance may also be initialized with the property values of a message. The mapping of message property to dataslot names can be specified in the start workstep using the following semantics.

\[
<\text{Copy to="@orderIdDS"}>orderId</\text{Copy}>
\]

In this mapping, the message property orderId points to an XPath (a location) in the message XML. The value of the oderid from that location in the message is copied to the dataslot orderIdDS. When mapping is not defined for a start workstep, the dataslots of the newly created instance are not initialized.

XML tags for start workstep

An example of how the XML tags are to be used for a start workstep is shown below.

```xml
<AtomicWS name="Start">
  <Priority>medium</Priority>
  <OnMessage>
    <Messages>
      <Message name="CreatePurchaseOrder">
        <MessageData>
          <Copy to="@SenderId">SenderId</Copy>
          <Copy to="@OrderId">orderId</Copy>
          <Copy to="@ItemName">item</Copy>
          <Copy to="@Quantity">quantity</Copy>
        </MessageData>
      </Message>
    </Messages>
  </OnMessage>
</AtomicWS>
```

In the above example, the first `<Copy>` tag copies the sender process instance ID from the JMS message header property SenderInstanceId to the dataslot SenderID. The remaining `<Copy>` tags copy the data from message properties orderId, item and quantity to the dataslots OrderID, ItemName and Quantity respectively.
Process instance creation using messages

When BP Server receives a message, the following steps are performed:

1. BP Server finds all the process templates whose Start workstep is subscribed to the received message.
2. For all such process templates found, a new process instance is created one by one.
3. The dataslots mapped to the message properties in the Copy operation are assigned corresponding values from the message.

Message flow diagram

Figure 34: Message Flow on page 215 shows the flow of the message.
Messaging example

Figure 35: Messaging example on page 216 shows how messaging works between multiple applications.

In the above figure, the application on the left hand side is ‘ProductManager’ and the application on the right hand side is ‘ProductApproval’. The publisher workstep—SendProductForApproval—in the ‘ProductManager’ application publishes a message with the following content:

```
<NewProduct>
  <ProductId>123</ProductId>
  <ProductName>Laptop</ProductName>
  <ProductQty>100</ProductQty>
  <ProductPrice>50000.00</ProductPrice>
  <ProductDescription>Laptop description here</ProductDescription>
</NewProduct>
```

This message also carries the “ApproverName” as the header property. When such a message is published, a new process instance of the second application ‘ProductApproval’ is created, and all data from this message is passed on to dataslots of the new process instance. The Header Property ‘ApproverName’ is also copied to a dataslot of the new process instance.

In the publisher workstep of second application, following message is sent:

```
<ProductApproval>
  <ID>123</ID>
  <Status>true</Status>
```
When such a message is published, it is correlated to the appropriate process instance of the first application ‘ProductManager’ based on ‘ID’. In this way, the requesting user comes to know about approver’s decision.
Using the BP Server admin utility

You can choose to publish Business Process Server applications using a command-oriented interface known as BP Server Admin. This section describes BP Server Admin commands and presents a sample session illustrating how to create and install process templates.

For details, see the following topics:

- Starting BP Server admin
- BP Server admin commands
- Refreshing configuration parameters
- Sample BP Server admin session

Starting BP Server admin

1. You can start the BP Server admin utility using the Windows Start menu:

   Start > All Programs > Progress > OpenEdge > BP Server > Admin Consoles > Business Process Console.

2. You can also start the BP Server admin utility from the command-line by going in the OEBPS_HOME\bin folder.

   startBPServerAdmin -u [username] -p [password] -f [file containing commands]

   where, the -f option can be used to provide the file containing the BP Server Admin commands to be executed with the call to the startBP ServerAdmin.
**BP Server admin commands**

For BP Server administration, you require only a few of the commands to complete several common administrative tasks, as listed in Table 21: BP Server admin commands on page 220. These commands work at various levels including the process, process template, and process instance.

**Note:** You must publish and/or refresh a BP Server application for the first time using Progress Developer Studio for OpenEdge or AppDeployer tool. Subsequently, you can perform these operations using BP Server admin utility. The commands are *not* case-sensitive.

### BP Server admin commands list

Table 21: BP Server admin commands

<table>
<thead>
<tr>
<th>To . . .</th>
<th>Use these commands . . .</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate and create a process instance from a specified XML file.</td>
<td>activate &lt;xmlFile&gt;</td>
</tr>
<tr>
<td>Reload the cache when a group is added in or removed from a group in the LDAP realm.</td>
<td>addOrRemoveGroupChildInCache &lt;parentGroupName&gt;&lt;childGroupName&gt;</td>
</tr>
<tr>
<td><strong>Note:</strong> In the LDAP realm, execute this command before adding a group to a group or removing a group from a group.</td>
<td></td>
</tr>
<tr>
<td>Reload the cache when a user is added to or removed from a group in the LDAP realm.</td>
<td>addOrRemoveUserChildInCache &lt;parentGroupName&gt;&lt;childUserName&gt;</td>
</tr>
<tr>
<td><strong>Note:</strong> In the LDAP realm, execute this command before adding a user to a group or removing a user from a group.</td>
<td></td>
</tr>
<tr>
<td>Add a new work item to the specified process instance for a specified workstep.</td>
<td>addWorkItem &lt;piName&gt; &lt;wsName&gt; &lt;wiPerformer&gt;</td>
</tr>
<tr>
<td>Assign a work item to a performer.</td>
<td>assignWi &lt;wiId&gt;</td>
</tr>
<tr>
<td>Assign a work item to a performer.</td>
<td>assign &lt;wiNo</td>
</tr>
<tr>
<td>To...</td>
<td>Use these commands...</td>
</tr>
<tr>
<td>-------</td>
<td>----------------------</td>
</tr>
<tr>
<td>Clear alert cache for the process template if a process name is specified. Alternatively, clear alert cache of all process templates.</td>
<td><code>clearAlertCache [ptName]</code>&lt;br&gt;Use this command to remove the alert metadata from the system memory. This forces BP Server to make database calls which can degrade its performance. After this command, depending on whether you have used enableAlertCache or disableAlertCache, it starts/stops caching.</td>
</tr>
<tr>
<td>Complete a work item with the specified work item ID.</td>
<td><code>completeWi &lt;wiId&gt;</code></td>
</tr>
<tr>
<td>Complete the work item.</td>
<td><code>complete &lt;wiNo&gt;</code></td>
</tr>
<tr>
<td>Connect to the BP Server.</td>
<td><code>connect &lt;userId&gt; &lt;password&gt;</code></td>
</tr>
<tr>
<td>Create a new version of the given template and deprecate the current version. Alternatively, create the first version.</td>
<td><code>createVersion &lt;xmlFile&gt;</code></td>
</tr>
<tr>
<td>Create a process from an XML file.</td>
<td><code>create &lt;xmlFile&gt;</code></td>
</tr>
<tr>
<td>Create and activate an instance.</td>
<td><code>activate &lt;ptName&gt; &lt;piName&gt; &lt;priority&gt;</code></td>
</tr>
<tr>
<td>Create and install a new version of an installed and running process.</td>
<td><code>createAndInstallPtVersion &lt;xml-fileName&gt; &lt;parentPtName&gt;</code></td>
</tr>
<tr>
<td>Create and install a process template.</td>
<td><code>createAndInstallPt &lt;ptName&gt;</code></td>
</tr>
<tr>
<td>Disable alert caching.</td>
<td><code>disableAlertCache</code>&lt;br&gt;Use this command to stop caching the alert metadata in the system memory. This forces BP Server to use the alert metadata cached before using this command.</td>
</tr>
<tr>
<td>Disconnect the BP Server.</td>
<td><code>disconnect</code></td>
</tr>
<tr>
<td>Enable alert caching.</td>
<td><code>enableAlertCache</code>&lt;br&gt;Use this command to start caching the alert metadata in the system memory. As BP Server uses the cached alert metadata, it does not have to make a database call. This improves the performance. Only BP Server can read the cached data.</td>
</tr>
<tr>
<td>Exit BP Server Admin.</td>
<td><code>exit</code></td>
</tr>
<tr>
<td>Retrieve and display all information about all alerts for the specified process template.</td>
<td><code>getAlertInfo &lt;ptName&gt;</code></td>
</tr>
<tr>
<td>Retrieve all active worksteps for the specified process template.</td>
<td><code>getAllWorksteps &lt;ptId&gt;</code></td>
</tr>
<tr>
<td><strong>To...</strong></td>
<td><strong>Use these commands...</strong></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>Retrieve a list of user authorized process templates, including the author of the process template.</td>
<td>getAuthPt</td>
</tr>
<tr>
<td></td>
<td>Process templates have a <strong>Group</strong> field. If <strong>Group</strong> is specified, then only members of that group are allowed to operate on that template. If <strong>Group</strong> is not specified, then all users are allowed to operate on that template.</td>
</tr>
<tr>
<td>Return the event ID of the last event processed by BPM Process Store.</td>
<td>getBizstoreEventCounter</td>
</tr>
<tr>
<td>Display the list of channels available on the server.</td>
<td>getChannels</td>
</tr>
<tr>
<td>Return comp-JavaScript functions for a specified workstep.</td>
<td>getCompFunction &lt;ptName&gt; &lt;wsName&gt;</td>
</tr>
<tr>
<td>Retrieve task data for the specified work item and specified dataslot name.</td>
<td>getData &lt;wiNo&gt; &lt;dsName&gt;</td>
</tr>
<tr>
<td>Get task data list for the specified work item</td>
<td>getDataList &lt;wiNo&gt;</td>
</tr>
<tr>
<td>Get the dataslot choice value for the specified process template or process instance.</td>
<td>getDataslotChoice &lt;ptName</td>
</tr>
<tr>
<td>Retrieve dataslot name, type, and value for the specified process template or process instance.</td>
<td>getDataslotInfo &lt;ptName</td>
</tr>
<tr>
<td>Get server information.</td>
<td>getInfo</td>
</tr>
<tr>
<td>Return all the worksteps within the InlineBlock for the specified process instance.</td>
<td>getInlineactivities &lt;piId&gt; &lt;InlineBlockName&gt;</td>
</tr>
<tr>
<td>Return the process manager name for the specified process template.</td>
<td>getManager &lt;ptName&gt;</td>
</tr>
<tr>
<td>Return the highest event ID generated by BP Server</td>
<td>getMaxEventId</td>
</tr>
<tr>
<td>Display the list of message names and description available on the server.</td>
<td>getMessages</td>
</tr>
<tr>
<td>Return all the missing events with the event ID greater than the specified event ID or all the missing events generated after the specified time.</td>
<td>getMissingEvents &lt;eventId</td>
</tr>
<tr>
<td>Display a list of all the process instances created by a user.</td>
<td>getOwnPi</td>
</tr>
<tr>
<td>To...</td>
<td>Use these commands...</td>
</tr>
<tr>
<td>-------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>Display the dataslot value associated with the specified process instance.</td>
<td>getPiData &lt;piId&gt; &lt;dsName&gt;</td>
</tr>
<tr>
<td>Return process instance due date.</td>
<td>getPiDueDate &lt;piName&gt;</td>
</tr>
</tbody>
</table>
| Display complete process instance information. | getPiInfo <piId>  
Or  
getPiStatus <piId> |
| Retrieve process instance’s active worksteps. | getPiWorkitem <piId> |
| Display process instance’s active worksteps. | getPiWorksteps <piId> |
| Get post-JavaScript functions for a specified workstep. | getPostFunction <ptName> <wsName> |
| Get precondition string for a specified workstep. | getPreCondition <ptName> <wsName> |
| Get pre-JavaScript functions for a specified workstep. | getPreFunction <ptName> <wsName> |
| Get all installed processes with appNames. | getProcessList  
Or  
getPiList |
| Get properties from bpserver.conf. | getProperties |
| Get process template’s AppName. | getPtAppName <ptName>  
Or  
getAppName <ptName> |
| Get process template’s dataslot value. | getPtData <ptName> <dsName> |
| Get process template’s information. | getPtInfo <ptName> |
| Display all currently installed process templates with details. | getPtList |
| Get all installed process template versions for the specified AppName. | getPTVersions <appName> |
| Display the list of MessagePublisher worksteps which publish the specified message. | getPublishers <messageName>  
If the message name is not specified, then all the MessagePublisher worksteps are displayed. |
To... | Use these commands...
---|---
Return workstep names to be reactivated for the specified error. | getReactivateWSName <ptName> <failureWs>
Return worksteps with rollback points. | getRollbackPoints <ptName>
Get the current state of the BP Server. | getServerState
Return the number of active sessions. | getSessionCount
Get the time the BP Server was started. | getStartTime
Return start workstep name of the specified process instance. | getStartWs <piId>
Displays the list of MessageSubscriber worksteps that have subscribed for the specified message name. | getSubscribers <messageName>
If the message name is not provided, then it displays all the MessageSubscriber worksteps.
Return a list of Suspended process templates. | getSuspendedPtList
Get Suspended worksteps of a process instance or process template. | getSuspendedWsList <ptName | piName>
Retrieve the work items assigned to the current user with their "available" status. | getTasks
Complete a workstep. | Enter getTasks and complete<wiNo> commands in that order.
Get all versions of the process template. | getVersions
Get the list of work items related to the specified process instance or workstep. | getWorkItemList <piId> <wsName>
Get the list of work items for the specified performer with their available status. | getWorkItems <performer>
Get the list of message instances to be delivered to the specified process instance ID and workstep name. | getWorkstepMessages <piId>
The process instance ID (piId) parameter is optional. If you do not provide this parameter, then BP Server displays message instances to be delivered to all process instances.
Return workstep due date. | getWSDueDate <piId> <wsName>
Install process template. | install <ptName>
The second of three stages for running a process. Validates and activates the process template. The first stage is creating a process and the third stage is activating a process.
<table>
<thead>
<tr>
<th>To...</th>
<th>Use these commands...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Check if the alert cache is enabled.</td>
<td><code>isAlertCacheEnabled</code></td>
</tr>
<tr>
<td>Check if the work item is assigned.</td>
<td><code>isAssigned &lt;wiNo&gt;</code></td>
</tr>
<tr>
<td>Check if the dynamic class loading is enabled.</td>
<td><code>isDynamicCL</code></td>
</tr>
<tr>
<td>Check if the process instance exists.</td>
<td><code>isPiExist &lt;piId&gt;</code></td>
</tr>
<tr>
<td>Check if the work item exists.</td>
<td><code>isWiExist &lt;wiId&gt;</code></td>
</tr>
</tbody>
</table>
| Load an external file from the file. | `loadscript <fileName>`  
Do not use the same name as the history file. |
| Make the specified work item available to the remaining performers who have not yet completed it, or to the specified performers. | `makeAvailable <wiNo> <performer | comma separated list of performers>` |
| Print the given message in the ejbserver console. | `printInfo <message>` |
| Restart the flow from the specified workstep. | `reactivateWs <piId> <wsName>` |
| Refresh the changes done in the callback XML configuration file without restarting the BP Server. | `refreshCallback` |
| Reload the JAR files from all the common paths as well as the process-template-specific path. | `refreshClassLoader` |
| Reload the JAR files from the process-template-specific path. | `refreshClassLoader <ptName>` |
| Reload the e-mail related properties from the `oebpsemail.properties` file, avoiding the need to restart BP Server. | `refreshEmail` |
| Reload the cached JMS resources. | `refreshJMS` |
| Reload Business Process Server properties and resources. | `refreshOEBPS` |
| Forcefully remove all the instances of the specified process template, remove the specified process template, and then create and install the specified process template. | `reinstallPt <ptName>` |
### Chapter 27: Using the BP Server admin utility

<table>
<thead>
<tr>
<th><strong>To...</strong></th>
<th><strong>Use these commands...</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Reload the configuration file for the server.</td>
<td>refreshBL</td>
</tr>
<tr>
<td>Remove all the instances of the specified process template.</td>
<td>removeAllPi &lt;ptName&gt;</td>
</tr>
</tbody>
</table>
| Remove all the process instances belonging to the specified process template. | removeAllPiLocal <ptName>  
**Note:** This does not remove the synchronous subprocess instances. |
| Remove all process templates and instances. | removeAllProcesses |
| Remove the specified group from the distributed GroupCache. | removeGroupFromCache <groupName>  
**Note:** In LDAP realm, execute this command before removing a group. |
| Remove the process instance from memory. | removePi <piId> |
| Remove a process template from memory. | removePt <ptName> |
| Remove reactivate workstep or reset reactivate attribute. | removeReactivateWsName <ptName> <wsName> |
| Remove the specified user from the distributed UserCache. | removeUserFromCache <userName>  
**Note:** In LDAP realm, execute this command before removing a user. |
<p>| Remove a work item. | removeWorkItem &lt;wiName|wiId&gt; |
| Refresh the process template without versioning. | replaceProcess |
| Resume all suspended adapters. | resumeallAdapterWs |
| Resume all suspended worksteps. | resumeallWs |
| Resume BP Server. | resumeBLServer &lt;username&gt; &lt;password&gt; |
| Resume client connections to the BP Server. | resumeConnections &lt;username&gt; &lt;password&gt; |
| Resume process instance. | resumePi &lt;piId&gt; |</p>
<table>
<thead>
<tr>
<th>To...</th>
<th>Use these commands...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resume Suspended process template.</td>
<td>resumePt &lt;ptName&gt;</td>
</tr>
<tr>
<td>Resume workstep from suspension due to JavaScript errors or external performer.</td>
<td>resumeWs &lt;piId&gt; &lt;wsName&gt;</td>
</tr>
<tr>
<td>Run the Garbage Collector forcibly.</td>
<td>runGC</td>
</tr>
<tr>
<td>Set activation time for a workstep in the specified format.</td>
<td>setActivationTime &lt;piId&gt; &lt;wsName&gt; &lt;long</td>
</tr>
<tr>
<td>Set comp-JavaScript function.</td>
<td>setCompFunction &lt;ptName&gt; &lt;wsName&gt; &lt;jsCode&gt; Use double quote for entire JavaScript code.</td>
</tr>
<tr>
<td>Set the debug flag to print the debug information about the fired commands in the bpserverclient.log file.</td>
<td>setDebug true/false</td>
</tr>
<tr>
<td>Set dynamic class loading.</td>
<td>setDynamicCL true/false</td>
</tr>
<tr>
<td>Set the process manager name.</td>
<td>setManager &lt;piName/ptName&gt; &lt;mgrName&gt;</td>
</tr>
<tr>
<td>Set process instance dataslot.</td>
<td>setPiData &lt;piId&gt; &lt;dsName&gt; &lt;data&gt;</td>
</tr>
<tr>
<td>Set the process instance due date.</td>
<td>setPiDueDate &lt;piId&gt; &lt;dueDate&gt;</td>
</tr>
<tr>
<td>Change the process instances priority.</td>
<td>setPiPriority &lt;piId&gt; &lt;priority&gt;</td>
</tr>
<tr>
<td>Set post-JavaScript functions for a workstep.</td>
<td>setPostFunction &lt;piName&gt; &lt;wsName&gt; &lt;jsCode&gt;</td>
</tr>
<tr>
<td>Set the precondition string.</td>
<td>setPreCondition &lt;ptName&gt; &lt;wsName&gt; &lt;preCondition&gt;</td>
</tr>
<tr>
<td>Set pre-JavaScript functions.</td>
<td>setPreFunction &lt;piName&gt; &lt;wsName&gt; &lt;jsCode&gt;</td>
</tr>
<tr>
<td>Set the process template data. This command is applicable only to the global dataslots.</td>
<td>setPtData &lt;ptName&gt; &lt;dsName&gt; &lt;data&gt;</td>
</tr>
<tr>
<td>Set reactivate workstep for worksteps.</td>
<td>setReactivateWsName &lt;ptName&gt; &lt;failureWs&gt;</td>
</tr>
<tr>
<td>Set dataslot names for worksteps when rollback occurs.</td>
<td>setSaveDataslots &lt;ptName&gt; &lt;wsName&gt; &lt;dsName_1&gt; &lt;dsName_2&gt;...&lt;dsName_n&gt;</td>
</tr>
<tr>
<td>Set the start workstep name.</td>
<td>setStartWs &lt;piId&gt; &lt;name&gt;</td>
</tr>
<tr>
<td>Set timer interval.</td>
<td>setTimer &lt;seconds&gt;</td>
</tr>
<tr>
<td>Set workstep due date.</td>
<td>setWsDueDate &lt;piId&gt; &lt;wsName&gt; &lt;dueDate&gt;</td>
</tr>
<tr>
<td><strong>To...</strong></td>
<td><strong>Use these commands...</strong></td>
</tr>
<tr>
<td>-----------------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>Set the datalot value as associated with the work item number.</td>
<td>setdata &lt;wiNo&gt; &lt;dsName&gt; &lt;data&gt;</td>
</tr>
<tr>
<td>Shutdown BP Server.</td>
<td>shutdown</td>
</tr>
<tr>
<td>Sleep for the specified milliseconds.</td>
<td>sleep &lt;milliSecs&gt;</td>
</tr>
<tr>
<td>Start the BP Server E-mail listener.</td>
<td>startEmailListener</td>
</tr>
</tbody>
</table>
| Start a process instance which is created, but not yet activated. | startProcessInstance <piId>  
A useful command if the BP Server is down—while in the middle of activating new process instances. |
| Start the BP Server. | startServer <userID> <password> |
| Stop the BP Server E-mail listener. | stopEmailListener |
| Suspend BP Server. | suspendBPServer |
| Suspend client connections to the BP Server | suspendConnections |
| Suspend a process instance. | suspendPi <piId> |
| Suspend a process template. | suspendPt <ptName>  
**Note:** The process template must be installed to suspend it successfully. |
| Suspend a workstep. | suspendWs |
| Print current time in the BP Server Admin console. | time |
| Update available work item’s performer. | updateAvailWiPerformer <wiNo> <performer> |
| Check if the process is valid for update. | validateProcessForUpdate |
| Get the version number of the BP Server. | version |

**Command parsing**

In pre-7.5 Business Process Server versions, BP Server Admin used to do command parsing by using the StringTokenizer with double quotes and space as the delimiters. Therefore, whenever a double quote was used in an argument, pre-7.5 BP Server Admin treated it as the end of that argument. From 7.5, you can use double quotes in an argument for a BP Server command by using back-slash (\) as an escape character.

For example, if you want to enter argument `jst.writeLog("Testing Log");`, then it should be entered as `jst.writeLog("\"Testing Log\"");`
The actual command would look like

```javascript
setPreFunction Approval NotifyApprove "jst.writeLog("Testing Log");"
```

The third argument would be taken as `jst.writeLog("Testing Log");`

Only the double quotes which are part of the argument must be escaped by a back-slash (`\`). Similarly, if you want to enter a multiline input, then you can do so using a back-slash (`\`) at the end of each line.

For example,

```javascript
setPreFunction Approval NotifyApprove "jst.writeLog("This is first line");"
```

```javascript
jst.writeLog("This is second line");
```

```javascript
jst.writeLog("This is second line");"
```

Here, the third argument would be taken as:

```javascript
jst.writeLog("This is first line");
jst.writeLog("This is second line");
jst.writeLog("This is third line");
```

### Using help for BP Server admin commands

BP Server Admin offers two levels of help.

- To see a list of available commands from the BP Server Admin window, enter:
  
  ```
  help
  ```

- To see more information about a particular command, enter the following:

  ```
  help <command_name>
  ```

### Debugging BP Server admin commands

You can enable debugging in BP Server admin utility to help you troubleshoot issues. To enable debugging, in the BP ServerClient category of the `OEBS_HOME\conf\oebpslog.conf` file, set the `log4j.category.BP ServerClient` parameter to debug, `BP ServerClient.File`. It appends the debug details in the `WORK_DIR\logs\bpserverclient.log` file.

### Refreshing configuration parameters

BP Server admin provides the following commands to refresh various configuration parameters.

### Refreshing BP Server properties and resources

The "refreshBL" command synchronizes the BP Server properties and resources in a single server. In the case of a single server, a synchronized call is made locally
This command refreshes the properties and resources by reading the following configuration files:

- `bpserver.conf`
- `oebps.conf`
- `oebpsdb.properties`
- `oebpsemail.properties`

### Refreshing JMS cached resources

The `refreshJMS` command synchronizes the JMS cache in a single server. In the case of a single server, a synchronized call is made locally.

This command cleans up all the references and JMS cache stored locally, and re-initializes the configuration information from `oebpsjndi.properties`.

### Refreshing Business Process Server properties and resources

The `refreshOEBPS` command synchronizes the Business Process Server properties and resources in a single server.

This command refreshes the properties and resources by reading the following configuration files:

- `oebps.conf`
- `oebpsdb.properties`
- `oebpsemail.properties`
- `oebpslog.conf`

### Broad outline for executing the commands

The following is a broad outline of steps for executing these commands:

1. Start BP Server.
2. Start BP ServerAdmin.
3. Connect to BP Server using a valid username and password.
4. Execute the command – `refreshBPS`, `refreshJMS` or `refreshOEBPS`.

You can check `bpserver.log` file for the results of these commands. The results of these respective commands are shown in Table 22: Log entries for refresh commands on page 230.

#### Table 22: Log entries for refresh commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Log entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>refreshBPS</td>
<td>BP Server INFO - (3222): Refresh of BP Server properties done successfully.</td>
</tr>
<tr>
<td>refreshJMS</td>
<td>BP Server INFO - (3221): JMS Resources cached in the server is refreshed successfully.</td>
</tr>
<tr>
<td>refreshOEBPS</td>
<td>OEBPS INFO - (029): Refresh of Business Process Server resources done successfully.</td>
</tr>
</tbody>
</table>
Sample BP Server admin session

The following illustrates a sample BP Server Admin session to create and install applications. Note that this session assumes that the BP Server is up and running—start the BP Server if it is not running.

1. Start BP Server Admin.
   - In Windows, choose **Start > All Programs > Progress > OpenEdge > BP Server > Admin Consoles > Business Process Console**.
   - In UNIX, enter `startBPServerAdmin.sh`.

2. Enter `connect admin admin`.
   - This connects you to the BP Server.

3. Create the process templates for one or more Business Process Server applications by entering the following:
   ```
   create Hiring.xml
   create Assignment.xml
   create approve.xml
   ```

4. Install the process templates by entering the following:
   ```
   install Hiring
   install Approval
   install Assignment
   ```

5. Enter exit.
   - When you log into your Home module, you should see the three new applications listed in the **Applications** page: Hiring, Approval, and Assignment.
This chapter explains what a monitoring process is and how Business Process Server handles it. A monitored process in the Business Process Server is a process that includes an external workstep (monitored workstep) containing a CHARACTER dataslot specified as the External Instance ID (EIID), and has all dataslots associated with external worksteps as editable.

For details, see the following topics:

- Overview
- External Instance Id
- Workstep transitions
- Supported events
- External event operations
- Multiple events in a single message
- Event processing
- Refreshing a Monitoring process
- Migrating a process instance of a monitoring process
Overview

OpenEdge BPM enables organizations to seamlessly automate processes, provides end-to-end visibility into process execution, and helps analyze them and identify areas for improvement. However, not all processes are automated using BPM platform. Many processes are already automated using custom or packaged applications like ERP, CRM, or legacy applications. Businesses need tools that enable Operational Responsiveness without the need to necessarily automate a process using BPM. A Monitoring Process uses the Business Events from an organization to "monitor" the actual business process on the BPM platform, irrespective of where and how the process is executed. Figure 36: Monitoring process on page 234 shows a schematic representation of monitoring process.

Figure 36: Monitoring process

The actual process can be executed on a packaged application, a legacy application, or it could be a semi-automated activity. However, as the monitoring process is executed and moves from one state to the next, Business Process Server can capture monitoring information, which provide an end-to-end visibility into the process for business users.
External Instance Id

Every monitoring process must have a user defined CHARACTER dataslot as an External Instance Id (EIId). The application developer needs to specify one of the user defined dataslots as an EIId. An EIId is an identifier used by external systems to identify a process instance in Business Process Server. EIId must be unique across all process instances of a process. Until the EIId is provided for the process instance, BP Server cannot process any external event of that instance.

Instead of providing an EIId, if required, the external system can use the process instance id as an EIId.

**Note:** If more than one EIIds are required for a monitoring process, then this requirement can be handled by using a subprocess.

---

Workstep transitions

Every monitoring workstep in a process goes through the following workstep transitions based on the events received.

W_CREATED -> W_EVENTACTIVATION_WAIT -> W_ACTIVATED -> W_COMPLETED

After creating a workstep, it goes into the W_EVENTACTIVATION_WAIT state and waits for the next event. It is then activated and completed depending on external events received for this workstep. You can change the state of a monitoring workstep using Business Process Portal or BP Server APIs.

---

Supported events

Business Process Server supports external events described in Table 23: Event types supported for a monitoring process on page 235 for a monitoring process.

**Table 23: Event types supported for a monitoring process**

<table>
<thead>
<tr>
<th>Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE_INSTANCE</td>
<td>This event triggers process instance creation in Business Process Server using the specified process name. The external process should pass EIId as part of this event, and this value must be unique across all process instances of a process. If the EIId is mapped to a process instance Id, then you should not pass the EIID. If you pass it, then BP Server ignores it, and logs a warning in the log file.</td>
</tr>
<tr>
<td>ACTIVATE_WORKSTEP</td>
<td>This event triggers workstep activation. The target workstep is identified using the EIId, process name, and workstep name value specified in this event. If this event contains any dataslot, then it is updated before activating the target workstep.</td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMPLETE WORKSTEP</td>
<td>This event triggers workstep completion. The target workstep is identified using the EIIId, process name, and workstep name values specified in this event. If this event contains any dataslot, then it is updated before completing the target workstep.</td>
</tr>
<tr>
<td>UPDATE_DATASLOT</td>
<td>This event updates the dataslot. The target process instance is identified using the EIIId, and process name specified in this event. BP Server processes this event as and when it is received.</td>
</tr>
</tbody>
</table>

In Progress Developer Studio for OpenEdge, you can configure ACTIVATE_WORKSTEP as mandatory for a monitoring process. Refer to the "Configuring monitoring workstep" section in *OpenEdge Getting Started: Developing BPM Applications with Developer Studio* for more information.

- If ACTIVATE_WORKSTEP event is mandatory for a monitoring process, then it will wait for ACTIVATE_WORKSTEP event, before proceeding with the COMPLETE_WORKSTEP event, even if it has arrived.
- If ACTIVATE_WORKSTEP event is not mandatory, then the workflow will process the event which comes first. For example, if COMPLETE_WORKSTEP event comes before ACTIVATE_WORKSTEP event, then it is processed and the ACTIVATE_WORKSTEP event is moved to invalid event table.

Alternatively, you can use the ABL APIs to work with the monitoring process in the Business Process Server. These ABL methods: StartProcess(), ActivateWorkstep(), CompleteWorkstep(), UpdateDataslots(), and GetProcess() enable you to work with monitoring process in the Business Process Server. These methods are public methods of the Progress.BPM.UserSession class. For more information on these ABL methods of the Progress.BPM.UserSession class, see the *OpenEdge Development: ABL Reference* guide.

**JMS event gateway**

The external system interacts with Business Process Server using event gateways. An event gateway ensures that, all the events received by it are persisted and processed. An event gateway supports JMS to accept external events.

The external system can send events as JMS messages to a pre-decided and pre-configured queue to create a monitoring process instance or to process the various steps involved as part of the monitoring process. These messages are processed by a MessageDrivenBean in Business Process Server.

Monitoring Process Queue is a persistent queue. Messages sent to this queue are persisted irrespective of the message delivery mode set by message sender (external system).

**Note:** To ensure that the messages are not lost even in the case of application server like JBoss, which does not support persistent JMS destination, Progress Software Corporation recommends that the external system must send all monitoring process JMS messages using persistent delivery mode.
**JMS message**

Any external event for a monitoring workstep is sent to Business Process Server as a JMS message. Business Process Server receives external events in the form of a JMS message for process instance creation, workstep activation, workstep completion, and dataslot update. These JMS messages have two parts, message properties and message body. The salient part of the event data is specified in the JMS message properties and the remaining data including dataslot(s) of a monitoring process or its worksteps is specified in the JMS message body. Business Process Server supports JMS messages of type `javax.jms.ObjectMessage` and `javax.jms.MapMessage`.

**Message properties**

BP Server reads the values of external instance id, process name, workstep name, and operation type from the JMS message properties. The message properties are same for both `javax.jms.ObjectMessage` and `javax.jms.MapMessage` JMS messages. The Table 24: JMS message properties details on page 237 explains the JMS message properties details.

**Table 24: JMS message properties details**

<table>
<thead>
<tr>
<th>Event element</th>
<th>Property name</th>
<th>Data type</th>
<th>Mandatory</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>External Instance Id</td>
<td>EXTERNAL_INSTANCE_ID</td>
<td>CHARACTER</td>
<td>Yes</td>
<td>A unique id to identify the target instance.</td>
</tr>
<tr>
<td>Process Name</td>
<td>PROCESS_NAME</td>
<td>CHARACTER</td>
<td>Yes</td>
<td>The process template name of the Business Process Server process.</td>
</tr>
<tr>
<td>Workstep Name</td>
<td>WORKSTEP_NAME</td>
<td>CHARACTER</td>
<td>Yes (for workstep events)</td>
<td>The workstep name for which the event is targeted. If operation type is ACTIVATE_WORKSTEP or COMPLETE_WORKSTEP, then it should have a proper value.</td>
</tr>
<tr>
<td>Operation Type</td>
<td>OPERATION_TYPE</td>
<td>CHARACTER</td>
<td>Yes</td>
<td>The operation to be performed in Business Process Server. Possible values: CREATE_INSTANCE, ACTIVATE_WORKSTEP, COMPLETE_WORKSTEP, UPDATE_DATASLOT</td>
</tr>
<tr>
<td>Instance Alias</td>
<td>INSTANCE_ALIAS</td>
<td>CHARACTER</td>
<td>No</td>
<td>The alias or prefix of the created instance name. If not provided, then process template name is the alias.</td>
</tr>
<tr>
<td>Email id</td>
<td>NOTIFY_EMAILS</td>
<td>CHARACTER</td>
<td>No</td>
<td>The comma separated e-mail addresses to which notification is to be sent about invalid messages.</td>
</tr>
</tbody>
</table>
Message body

BP Server reads the values of sender, create time, and other optional event elements from the JMS message body. The Table 25: JMS ObjectMessage body details on page 238 explains the JMS message body details of ObjectMessage. For ObjectMessage, the message body must be an instance of `java.util.HashMap`. The 'key' for the HashMap must be of type `java.lang.String`. You can find a list of valid keys in the 'Property name' column of the Table 25: JMS ObjectMessage body details on page 238. For each entry in the HashMap, the 'value' must be of the corresponding type specified in the 'Data type' column.

Table 25: JMS ObjectMessage body details

<table>
<thead>
<tr>
<th>Event element</th>
<th>Property name</th>
<th>Data type</th>
<th>Mandatory</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sender</td>
<td>SENDER</td>
<td>CHARACTER</td>
<td>Yes</td>
<td>The external system information from where the event is generated.</td>
</tr>
<tr>
<td>Category</td>
<td>CATEGORY</td>
<td>CHARACTER</td>
<td>No</td>
<td>This can be used by senders to group events based on functional domains. For example, Finance, CRM.</td>
</tr>
<tr>
<td>Priority</td>
<td>PRIORITY</td>
<td>CHARACTER</td>
<td>No</td>
<td>This is the priority of the operation performed in the external system. This is updated in the process flow. Possible values: CRITICAL, HIGH, MEDIUM, LOW. If not specified, then the default value &quot;MEDIUM&quot; is used.</td>
</tr>
<tr>
<td>Performer</td>
<td>PERFORMER</td>
<td>CHARACTER</td>
<td>No</td>
<td>Considered as the instance creator. If not provided, then the BP Server bootstrap user specified during Business Process Server installation is used. Workstep events: The value is considered as the workstep performer. If not provided, then the ‘default performer’ is used. Other events: This value, if provided, in any other event is ignored.</td>
</tr>
<tr>
<td>Loop counter</td>
<td>LOOP_COUNTER</td>
<td>INTEGER</td>
<td>No (may be provided for loops)</td>
<td>The counter for loops or iterations. If not specified, then events cannot be processed in order. This value is used only in workstep events. It is ignored for other events.</td>
</tr>
<tr>
<td>Create Time</td>
<td>CREATE_TIME</td>
<td>INT64</td>
<td>Yes</td>
<td>This is the current time in the external system when the event was generated. The value should be specified in milliseconds.</td>
</tr>
<tr>
<td>Event element</td>
<td>Property name</td>
<td>Data type</td>
<td>Mandatory</td>
<td>Comment</td>
</tr>
<tr>
<td>---------------</td>
<td>---------------</td>
<td>-------------</td>
<td>-----------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Start time</td>
<td>START_TIME</td>
<td>INT64</td>
<td>No</td>
<td>This is the workstep or process instance start time. The value should be specified in milliseconds. This value is used only in CREATE_INSTANCE, ACTIVATE_WORKSTEP and COMPLETE_WORKSTEP events, and it is ignored for all other events.</td>
</tr>
<tr>
<td>End time</td>
<td>END_TIME</td>
<td>INT64</td>
<td>No</td>
<td>This is the workstep or process instance end time. The value should be specified in milliseconds. This value is used only in COMPLETE_WORKSTEP event, and it is ignored for all other events.</td>
</tr>
<tr>
<td>Duration</td>
<td>DURATION</td>
<td>INT64</td>
<td>No</td>
<td>This is the workstep duration. The value should be specified in seconds. This value is used only in ACTIVATE_WORKSTEP event and COMPLETE_WORKSTEP event, and it is ignored for all other events. If it is passed as ACTIVATE_WORKSTEP event, then it is taken as estimated duration, whereas if it is passed as COMPLETE_WORKSTEP event, then it is taken as actual duration.</td>
</tr>
<tr>
<td>Context</td>
<td>CONTEXT</td>
<td>HashMap</td>
<td>No</td>
<td>The external system can add any extra data to the context. To send such data, prepare a HashMap&lt;String,String&gt;, where 'key' and 'value' are of type java.lang.String.</td>
</tr>
<tr>
<td>Dataslot</td>
<td>DATASLOT</td>
<td>HashMap</td>
<td>No</td>
<td>The dataslot to be updated in the process flow. The key is the dataslot name and value is the dataslot value. For more information about passing it as part of event, refer to Dataslot update on page 245.</td>
</tr>
</tbody>
</table>

The Table 26: JMS MapMessage body details on page 239 explains the JMS message body details of MapMessage. The 'name' used in the MapMessage must be of type java.lang.String. You can find a list of valid names in the 'Property name' column of the Table 26: JMS MapMessage body details on page 239. For each name in the MapMessage, the 'value' must be of the corresponding type specified in the 'Data type' column.

Table 26: JMS MapMessage body details

<table>
<thead>
<tr>
<th>Event element</th>
<th>Property name</th>
<th>Data type</th>
<th>Mandatory</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sender</td>
<td>SENDER</td>
<td>CHARACTER</td>
<td>Yes</td>
<td>The external system information from where the event is generated.</td>
</tr>
<tr>
<td>Event element</td>
<td>Property name</td>
<td>Data type</td>
<td>Mandatory</td>
<td>Comment</td>
</tr>
<tr>
<td>---------------</td>
<td>---------------</td>
<td>-----------</td>
<td>-----------</td>
<td>---------</td>
</tr>
<tr>
<td>Category</td>
<td>CATEGORY</td>
<td>CHARACTER</td>
<td>No</td>
<td>This can be used by senders to group events based on functional domains. For example, Finance, CRM.</td>
</tr>
<tr>
<td>Priority</td>
<td>PRIORITY</td>
<td>CHARACTER</td>
<td>No</td>
<td>This is the priority of the operation performed in the external system. This is updated in the process flow. Possible values: CRITICAL, HIGH, MEDIUM, LOW. If not specified, then the default value &quot;MEDIUM&quot; is used.</td>
</tr>
<tr>
<td>Performer</td>
<td>PERFORMER</td>
<td>CHARACTER</td>
<td>No</td>
<td>Considered as the instance creator. If not provided, then the BP Server bootstrap user specified during Business Process Server installation is used. Workstep events: The value is considered as the workstep performer. If not provided, then the ‘default performer’ is used. Other events: This value, if provided, in any other event is ignored.</td>
</tr>
<tr>
<td>Loop counter</td>
<td>LOOP_COUNTER</td>
<td>INTEGER, INT64</td>
<td>No (may be provided for loops)</td>
<td>The counter for loops or iterations. If not specified, then events cannot be processed in order. This value is used only in workstep events. It is ignored for other events.</td>
</tr>
<tr>
<td>Create Time</td>
<td>CREATE_TIME</td>
<td>INTEGER, INT64</td>
<td>Yes</td>
<td>This is the current time in the external system when the event was generated. The value should be specified in milliseconds.</td>
</tr>
<tr>
<td>Start time</td>
<td>START_TIME</td>
<td>INTEGER, INT64</td>
<td>No</td>
<td>This is the workstep or process instance start time. The value should be specified in milliseconds. This value is used only in CREATE_INSTANCE, ACTIVATE_WORKSTEP and COMPLETE_WORKSTEP events, and it is ignored for all other events.</td>
</tr>
<tr>
<td>End time</td>
<td>END_TIME</td>
<td>INTEGER, INT64</td>
<td>No</td>
<td>This is the workstep or process instance end time. The value should be specified in milliseconds. This value is used only in COMPLETE_WORKSTEP event, and it is ignored for all other events.</td>
</tr>
</tbody>
</table>
**Duration**

**Property name**: DURATION  
**Data type**: INTEGER, INT64  
**Mandatory**: No  
**Comment**: This is the workstep duration. The value should be specified in seconds. This value is used only in ACTIVATE_WORKSTEP event and COMPLETE_WORKSTEP event, and it is ignored for all other events. If it is passed as ACTIVATE_WORKSTEP event, then it is taken as estimated duration, whereas if it is passed as COMPLETE_WORKSTEP event, then it is taken as actual duration.

**Context**

**Property name**: CONTEXT  
**Data type**: RAW  
**Mandatory**: No  
**Comment**: The external system can add any extra data to the context. To send such data, prepare a HashMap<String, String>, where 'key' and 'value' are of type java.lang.String. Convert this HashMap to byte[], and set it in the Map message body.

**Dataslot**

**Property name**: DATASLOT  
**Data type**: RAW  
**Mandatory**: No  
**Comment**: The dataslot to be updated in the process flow. To send dataslot(s), prepare a HashMap<String, Object>, where 'key' is dataslot name and 'value' is the dataslot value. Convert this HashMap to byte[], and set it in the Map message body. For more information about passing it as part of event, refer toDataslot update on page 245.

### Connecting to a JMS event gateway

You must connect to a JMS event gateway to send an event to Business Process Server. All the JMS messages (events) should be delivered to the destination BMExternalEventQueue using the connection factory BMConnectionFactoryXA. The details for connecting to a JMS event gateway are available in the oebpsjndi.properties file present in OEBPS_HOME\conf folder. The Table 27: JMS event gateway details on page 241 lists the parameters you must check from the oebpsjndi.properties file.

#### Table 27: JMS event gateway details

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>oebps.&lt;appserver&gt;.jms.factory.initial</td>
<td>Initial context factory class name.</td>
</tr>
<tr>
<td>oebps.&lt;appserver&gt;.jms.provider.url</td>
<td>Provider URL.</td>
</tr>
<tr>
<td>oebps.&lt;appserver&gt;.jms.principal</td>
<td>User name.</td>
</tr>
<tr>
<td>oebps.&lt;appserver&gt;.jms.credentials</td>
<td>Password.</td>
</tr>
</tbody>
</table>
External event operations

Business Process Server supports the following external event operations using JMS message.

- Event to create a monitoring process instance.
- Event to activate a monitoring workstep of a monitoring process instance.
- Event to complete a monitoring workstep of a monitoring process instance.
- Event to update the dataslot(s) of a monitoring process instance.
- Event to create a non-monitoring process instance.

Process instance creation

An external CREATE_INSTANCE event triggers process instance creation in Business Process Server. This event must contain the External Instance Id (EIlid) value. Business Process Server validates the EIlid value for uniqueness. Refer to External event operations on page 242. Business Process Server reads the following data from this event associated with the instance.

- External Instance Id — This event must contain the EIlid value. This value is validated for uniqueness and is assigned to the EIlid dataslot in the Monitoring page in the Properties page of the monitoring process.
- Start time — If the external CREATE_INSTANCE event contains start time, then that start time is used; otherwise the external event create time is used as the start time. This start time is used as the instance creation time.
- Creator — The external CREATE_INSTANCE event may contain the value for performer. This performer value is used as the creator of the process instance. This performer must not be a valid Business Process Server user. If the external event does not contain a performer, then the Business Process Server user specified during Business Process Server installation is considered as the creator.
- Dataslots — Dataslots received from the external event, which are supported and are defined in the target process, can be passed with the external CREATE_INSTANCE event, and BP Server processes those dataslots. Invalid dataslots contained in the event are ignored. For more details about the supported dataslots, refer to Dataslot update on page 245.

Process instance creation for non-monitoring external process

In general all processes are of non-monitoring type. Only when the process needs to be controlled by an external system then it can be made monitoring. The life-cycle of monitoring worksteps in a process can be controlled by users in an external system using events. An external process can create an instance of a non-monitoring process by sending events. Once an instance of a non-monitoring process is created, thereafter, it is controlled by OpenEdge API or Business Process Portal.

Note: For non-monitoring external processes, you must complete the following events using Business Process Portal or BP Server APIs:

- ACTIVATE_WORKSTEP
- COMPLETE_WORKSTEP
- UPDATE_DATASLOT
The elements (properties and message body) of the event to be sent as part of JMS message is listed in the following Table 28: JMS MapMessage properties - non-monitoring process on page 243. Both JMS MapMessage and ObjectMessage are supported and the steps to create the message are similar to those to create a CREATE_INSTANCE event for a monitoring process.

Note: The following fields are not considered when creating non-monitoring external process:

- **Start time**: For a non-monitoring process, the process instance start time of event is not considered, even if it is passed. Only the time when the instance is getting created on OpenEdge is considered.
- **Additional data in event**: Any additional data (other than the event elements provided in the table) provided as part of the event is ignored. For example, External Instance Id is not required, because once the instance is created, other events cannot trigger operations on this instance.

### JMS message properties

The following Table 28: JMS MapMessage properties - non-monitoring process on page 243 explains the JMS message properties.

#### Table 28: JMS MapMessage properties - non-monitoring process

<table>
<thead>
<tr>
<th>Event element</th>
<th>Property name</th>
<th>Data type</th>
<th>Mandatory</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process Name</td>
<td>PROCESS_NAME</td>
<td>CHARACTER</td>
<td>Yes</td>
<td>The process template name of the Business Process Server process.</td>
</tr>
<tr>
<td>Operation Type</td>
<td>OPERATION_TYPE</td>
<td>CHARACTER</td>
<td>Yes</td>
<td>The operation to be performed in Business Process Server. Possible value: CREATE_INSTANCE</td>
</tr>
<tr>
<td>Instance Alias</td>
<td>INSTANCE_ALIAS</td>
<td>CHARACTER</td>
<td>No</td>
<td>The alias or prefix of the created instance name. If not provided, then process template name is the alias.</td>
</tr>
<tr>
<td>Email id</td>
<td>NOTIFY_EMAILS</td>
<td>CHARACTER</td>
<td>No</td>
<td>The comma separated e-mail addresses to which notification is to be sent about invalid messages.</td>
</tr>
</tbody>
</table>

### JMS message body

The following Table 29: JMS ObjectMessage body - non-monitoring process on page 243 explains the JMS message body.

#### Table 29: JMS ObjectMessage body - non-monitoring process

<table>
<thead>
<tr>
<th>Event element</th>
<th>Property name</th>
<th>Data type</th>
<th>Mandatory</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sender</td>
<td>SENDER</td>
<td>CHARACTER</td>
<td>Yes</td>
<td>The external system information from where the event is generated.</td>
</tr>
<tr>
<td>Category</td>
<td>CATEGORY</td>
<td>CHARACTER</td>
<td>No</td>
<td>This can be used by senders to group events based on functional domains. For example, Finance, CRM.</td>
</tr>
<tr>
<td>Priority</td>
<td>PRIORITY</td>
<td>CHARACTER</td>
<td>No</td>
<td>This is the priority of the instance to be created. Possible values: CRITICAL, HIGH, MEDIUM, LOW. The default is MEDIUM.</td>
</tr>
<tr>
<td>Event element</td>
<td>Property name</td>
<td>Data type</td>
<td>Mandatory</td>
<td>Comment</td>
</tr>
<tr>
<td>---------------</td>
<td>---------------</td>
<td>----------------------</td>
<td>-----------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Performer</td>
<td>PERFORMER</td>
<td>CHARACTER</td>
<td>No</td>
<td>Considered as the instance creator. If not provided, then the BP Server bootstrap user specified during Business Process Server installation is used.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>Note:</strong> If the performer specified is an invalid BP Server user, then no BP Server user can see this instance in their MyInstance tab.</td>
</tr>
<tr>
<td>Create Time</td>
<td>CREATE_TIME</td>
<td>INT64</td>
<td>Yes</td>
<td>This is the current time in the external system when the event was generated. The value should be specified in milliseconds.</td>
</tr>
<tr>
<td>Context</td>
<td>CONTEXT</td>
<td>HashMap object</td>
<td>No</td>
<td>The external system can add any extra data to the context. To send such data, prepare a HashMap&lt;String,String&gt;, where key and value are of type java.lang.String.</td>
</tr>
<tr>
<td>Dataslot</td>
<td>DATASLOT</td>
<td>HashMap object</td>
<td>No</td>
<td>The dataslot to be updated in the process flow. The key is the dataslot name and value is the dataslot value. For more information about passing a dataslot as part of event, refer to Dataslot update on page 245.</td>
</tr>
</tbody>
</table>

**Workstep activation and completion**

The external events supported for workstep are - external ACTIVATE_WORKSTEP event and external COMPLETE_WORKSTEP event. Business Process Server reads the following data from this event associated with the workstep.

- Deriving start time — BP Server uses the start time from the external ACTIVATE_WORKSTEP event. If ACTIVATE_WORKSTEP event does not contain start time, then event creation time is taken as the start time. However, if only COMPLETE_WORKSTEP event is received, then it reads the start time from the external COMPLETE_WORKSTEP event. If ACTIVATE_WORKSTEP event is not received and start time is not available in the COMPLETE_WORKSTEP event, then BP Server derives it using the following logic:
  - If the COMPLETE_WORKSTEP event contains completion time and duration, then the start time is "end time - duration".
  - If the COMPLETE_WORKSTEP event contains end time, but does not contain duration, then the start time is "end time - estimated duration".
• If the COMPLETE_WORKSTEP event does not contain end time and duration, then end event create time is considered as the end time. Therefore, the start time is "end time - estimated duration".

• Deriving End time — The end time is taken only from the COMPLETE_WORKSTEP event; if it is specified in the ACTIVATE_WORKSTEP event, then BP Server ignores it. If the end time is not specified in the event, then BP Server derives it using the following logic:
  • If the ACTIVATE_WORKSTEP event contains start time, and COMPLETE_WORKSTEP event contains duration, then end time is "start time + duration".
  • If the ACTIVATE_WORKSTEP event contains start time, but duration is not provided in the COMPLETE_WORKSTEP event, then end time is "start time + estimated duration".
  • If ACTIVATE_WORKSTEP event does not contain the start time, and COMPLETE_WORKSTEP event does not contain the end time and duration, then COMPLETE_WORKSTEP event create time is considered as the end time.

• Deriving Duration — If the duration (in seconds) is provided in the COMPLETE_WORKSTEP event, then it takes precedence over the calculated value (workstep end time - workstep start time). If the provided duration is greater than the calculated value, then BP Server logs a warning in the log file and uses the provided duration. If the event does not contain start time, end time, and duration, then Business Process Server's workstep estimated duration is considered as the duration.

• Performer — The performer is mandatory for human worksteps. The performer need not be a Business Process Server user. For a monitoring human workstep, application developer can specify a single user or a mapped dataslot as the "default performer" while designing a process. If the external ACTIVATE_WORKSTEP event for human workstep does not contain performer, then default performer is used. If the default performer is invalid, then the Business Process Server user specified during Business Process Server installation is used.

• Loop counter — If the monitoring workstep is executed within a loop construct, then the external application needs to provide the loop counter as part of workstep events to handle out-of-sequence events. Without the loop counter, BP Server may not be able to process the events of a workstep in a loop. The loop counter starts from 1.

• Dataslots — For an external workstep event (ACTIVATE_WORKSTEP or COMPLETE_WORKSTEP), BP Server processes only the dataslots defined in that workstep, it ignores the remaining dataslots. For the list of supported dataslots, refer to Dataslot update on page 245. If EIIId dataslot is passed as part of dataslots, then BP Server ignores it and adds a warning in the log.

**Dataslot update**

A monitoring process can contain dataslots of any type, but BP Server supports only a certain dataslots for monitoring worksteps. The Table 30: Java types for monitoring process dataslots on page 245 describes the java type for each of the supported dataslot in a monitoring process.

**Table 30: Java types for monitoring process dataslots**

<table>
<thead>
<tr>
<th>Dataslot type</th>
<th>Java type</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT64</td>
<td>com.progress.lang.Int64</td>
<td></td>
</tr>
<tr>
<td>INTEGER</td>
<td>com.progress.lang.Integer</td>
<td></td>
</tr>
<tr>
<td>Dataslot type</td>
<td>Java type</td>
<td>Remarks</td>
</tr>
<tr>
<td>-----------------</td>
<td>----------------------------------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CHARACTER</td>
<td>com.progress.lang.Character</td>
<td>If the CHARACTER dataslot length is longer than the maximum length specified in the process template, then the excess length is truncated.</td>
</tr>
<tr>
<td>DECIMAL</td>
<td>com.progress.lang.Decimal</td>
<td></td>
</tr>
<tr>
<td>HANDLE</td>
<td>com.progress.open4gl.Handle</td>
<td></td>
</tr>
<tr>
<td>LOGICAL</td>
<td>com.progress.lang.Logical</td>
<td></td>
</tr>
<tr>
<td>DATETIME-TZ</td>
<td>java.lang.GregorianCalendar</td>
<td>In milliseconds.</td>
</tr>
<tr>
<td>LIST</td>
<td>java.util.Vector</td>
<td></td>
</tr>
<tr>
<td>MAP</td>
<td>java.util.Hashmap</td>
<td></td>
</tr>
<tr>
<td>OBJECT/BUSINESS</td>
<td>---</td>
<td>Not supported</td>
</tr>
<tr>
<td>OBJECT/ BUSINESS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OBJECT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MEMPTR</td>
<td>com.progress.open4gl.Memptr</td>
<td></td>
</tr>
<tr>
<td>TABLE</td>
<td>com.progress.open4gl.ProDatagraph</td>
<td></td>
</tr>
<tr>
<td>DATASET</td>
<td>com.progress.open4gl.ProDatagraph</td>
<td></td>
</tr>
</tbody>
</table>

The external UPDATE_DATASLOT event can contain any of the supported dataslots defined in the target process. BP Server processes only the valid dataslots and ignores the remaining dataslots. If EIId dataslot is passed as part of dataslots, then BP Server ignores it and adds a warning in the log.

**Loop**

Using loop or iterator in a monitoring process is not recommended. However, if you have to use a loop, then you can choose one of the following options.

- The external system should provide the loop counter along with the event. This helps BP Server in identifying the current loop count, and handle out-of-sequence events.

- If the external system does not have the information about loop counter, then there is a possibility of missing or step-over of certain events. If this behavior is acceptable, then you can choose this option.

**Subprocess**

A subprocess (which is also an independent process) has its own EIId, which is specified by the parent process or updated later in the subprocess instance. The parent process should receive this value in any workstep prior to subprocess. The parent process can pass its EIId to the child process, if required, with proper mappings. For example, the parent process can pass the OrderId and ShipmentId to the subprocess, which can be used by the subprocess to identify the order to which this shipment belongs to.
Multiple events in a single message

In pre-11.6 version of BP Server, each monitoring process message received by the JMS event gateway from an external system can contain only one event. However, current version of BP Server supports handling multiple or bulk events per JMS message. Every JMS message can contain one or more events. BP Server supports multiple or bulk events per JMS message for messages of type ObjectMessage, and MapMessage.

ObjectMessage with multiple events

If you want to send a single event in an ObjectMessage, then you must create a Map object and set it to the ObjectMessage. If you want to send multiple or bulk events as part of a single message, then the ObjectMessage must contain an object of type List. Every element in this List must be an event, which will be a Map.

To send multiple or bulk events as part of one ObjectMessage:

1. Create a Map for each message containing the event payload. Refer to Table 31: Map of event payload (data) for ObjectMessage and MapMessage on page 247 for more details about event payload properties.

Table 31: Map of event payload (data) for ObjectMessage and MapMessage

<table>
<thead>
<tr>
<th>Event element</th>
<th>Property name</th>
<th>Data type</th>
<th>Mandatory</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sender</td>
<td>SENDER</td>
<td>CHARACTER</td>
<td>Yes</td>
<td>The external system information which generates the event.</td>
</tr>
<tr>
<td>Category</td>
<td>CATEGORY</td>
<td>CHARACTER</td>
<td>No</td>
<td>This can be used by senders to group events based on functional domains. For example, Finance, CRM.</td>
</tr>
<tr>
<td>Priority</td>
<td>PRIORITY</td>
<td>CHARACTER</td>
<td>No</td>
<td>This is the priority of the operation performed in the external system. This is updated in the process flow. Possible values are CRITICAL, HIGH, MEDIUM, LOW. If not specified, then the default value MEDIUM is used.</td>
</tr>
<tr>
<td>Performer</td>
<td>PERFORMER</td>
<td>CHARACTER</td>
<td>No</td>
<td>This is considered as the instance creator. If not provided, then the BP Server bootstrap user specified during Business Process Server installation is used. For workstep events, the value is considered as the workstep performer. If not provided, then the default performer is used. For other events, the value, if provided, is ignored.</td>
</tr>
<tr>
<td>Loop counter</td>
<td>LOOP_COUNTER</td>
<td>INTEGER</td>
<td>No</td>
<td>The counter for loops or iterations. If not specified, then events cannot be processed in order. This value is used only in workstep events. It is ignored for other events.</td>
</tr>
<tr>
<td>Event element</td>
<td>Property name</td>
<td>Data type</td>
<td>Mandatory</td>
<td>Comment</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------------</td>
<td>-----------------</td>
<td>-----------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Create time</td>
<td>CREATE_TIME</td>
<td>DATETIME-TZ</td>
<td>Yes</td>
<td>This is the current time in the external system when the event was generated. The value is specified in milliseconds.</td>
</tr>
<tr>
<td>Start time</td>
<td>START_TIME</td>
<td>DATETIME-TZ</td>
<td>No</td>
<td>This is the workstep or process instance start time. The value is specified in milliseconds. This value is used only in CREATE_INSTANCE, ACTIVATE_WORKSTEP, and COMPLETE_WORKSTEP events. It is ignored for all other events.</td>
</tr>
<tr>
<td>End time</td>
<td>END_TIME</td>
<td>DATETIME-TZ</td>
<td>No</td>
<td>This is the workstep or process instance end time. The value is specified in milliseconds. This value is used only in COMPLETE_WORKSTEP event. It is ignored for all other events.</td>
</tr>
<tr>
<td>Duration</td>
<td>DURATION</td>
<td>INT64</td>
<td>No</td>
<td>This is the workstep duration. The value is specified in seconds. This value is used only in ACTIVATE_WORKSTEP, and COMPLETE_WORKSTEP event. It is ignored for all other events. If it is passed as ACTIVATE_WORKSTEP event, then it is taken as estimated duration, whereas if it is passed as COMPLETE_WORKSTEP event, then it is taken as actual duration.</td>
</tr>
<tr>
<td>Context</td>
<td>CONTEXT</td>
<td>HashMap object</td>
<td>No</td>
<td>The external system can add any extra data to the context. To send such data, prepare a HashMap&lt;String,String&gt;, where 'key' and 'value' are of type java.lang.String.</td>
</tr>
<tr>
<td>Dataslot</td>
<td>DATASLOT</td>
<td>HashMap object</td>
<td>No</td>
<td>The dataslot to be updated in the process flow. The key is the dataslot name, and value is the dataslot value.</td>
</tr>
</tbody>
</table>

2. Create another Map for event properties. Refer to Table 32: Map of event properties (header) for ObjectMessage and MapMessage on page 248 for more details about event properties.

<table>
<thead>
<tr>
<th>Event element</th>
<th>Property name</th>
<th>Data type</th>
<th>Mandatory</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>External Instance Id</td>
<td>EXTERNAL_INSTANCE_ID</td>
<td>CHARACTER</td>
<td>Yes</td>
<td>A unique id to identify the target instance.</td>
</tr>
<tr>
<td>Process Name</td>
<td>PROCESS_NAME</td>
<td>CHARACTER</td>
<td>Yes</td>
<td>The process template name of the Business Process Server process.</td>
</tr>
</tbody>
</table>
### Event element | Property name | Data type | Mandatory | Comment
--- | --- | --- | --- | ---
Workstep Name | WORKSTEP_NAME | CHARACTER | Yes (for workstep events) | The workstep name for which the event is targeted. If operation type is ACTIVATE_WORKSTEP or COMPLETE_WORKSTEP, then it should have a proper value.
Operation Type | OPERATION_TYPE | CHARACTER | Yes | The operation to be performed in Business Process Server. Possible values: CREATE_INSTANCE, ACTIVATE_WORKSTEP, COMPLETE_WORKSTEP, UPDATE_DATASLOT
Instance Alias | INSTANCE_ALIAS | CHARACTER | No | The alias or prefix of the created instance name. If not provided, then process template name is the alias.
Email id | NOTIFY_EMAILS | CHARACTER | No | The comma separated e-mail addresses to which notification is to be sent about invalid messages.

3. Add the payload (data) Map and header (properties) Map to a Map with keys EVENT_DATA and EVENT_PROPERTIES keys respectively.

4. Add all these event Maps containing header and payload (one Map per event) to a list.

5. Set this list on the ObjectMessage. The List object must be of type List<Map<String,Map<String,Object>>>.

6. Set the boolean property BULK_MESSAGE with value as 'true' or 'false' for this message. If BP Server does not find this property or if its value is set to 'false', then BP Server handles it as a single event message otherwise it handles it as a multi-event message.

### MapMessage with multiple events

If you want to send a single event in a MapMessage, then you must add all the data elements to the MapMessage with pre-decided keys. If you want to send multiple or bulk events as part of a single message, then the message must contain one key containing a byte array value. The byte array will be a List<Map<String,Map<String,Object>>> as mentioned in ObjectMessage with multiple events on page 247.

**To send multiple or bulk events as part of one MapMessage:**

1. Create a Map for each message containing the event payload. Refer to Table 31: Map of event payload (data) for ObjectMessage and MapMessage on page 247 for more details about event payload properties.

2. Create another Map for event properties. Refer to Table 32: Map of event properties (header) for ObjectMessage and MapMessage on page 248 for more details about event properties.

3. Add the payload (data) Map and header (properties) Map to a Map with keys EVENT_DATA and EVENT_PROPERTIES keys respectively.

4. Add all these event Maps containing header and payload (one Map per event) to a List.
5. Put this List on the MapMessage as bytes (using `setBytes()`) with key as `EVENT_LIST`. The byte array in the message will be an object of type `List<Map<String,Map<String,Object>>>`.

6. Set the boolean property `BULK_MESSAGE` with value as `true` or `false` for this message. If BP Server does not find this property or if its value is set to `false`, then BP Server handles it as a single event message otherwise it handles it as a multi-event message.

**Processing of messages with multiple events**

BP Server separates the ObjectMessages and MapMessages received by the JMS event gateway as single event message or multiple event message based on the property `BULK_MESSAGE`. If it is a multiple event message, then BP Server processes every event one after another in a sequential order. Once individual events are extracted from the multiple event message, BP Server processes the message like a single event message. All the validations associated with a single event are applicable to all the events in the multiple event message also.

**BP Server processes a multiple event message as follows.**

1. Check whether the bulk message format is valid or not.
   - If the bulk message format is invalid, move the message to the `BIZEVENT_EXTERNAL_INVALID` table.
   - If the bulk message format is valid, extract individual events from the bulk message, and process one event at a time, one after the other in a sequential order.

2. Check whether the event is valid or not.
   - If the event is invalid, add it to the `BIZEVENT_EXTERNAL_INVALID` table.
   - If the event is valid, the JMS message gateway processes it. If the JMS message gateway cannot process it, it stores that event in the `BIZEVENT_EXTERNAL` table.

3. After all the valid events of a message are processes, if you have configured BP Server to save them in the history table, BP Server marks them as processes, and after a predefined period, moves them to history table in bulk.

**Note:** By default, BP Server does not store processes external events in the history table. To persist the processed events in the `BIZEVENT_EXTERNAL_HISTORY` table, set the `smp.enable.extevent.history` parameter in the `smp.properties` file to `true`.

**Event processing**

The JMS event gateway persists the events in persistent store when it receives them. Each event is stored in the order in which it is received. The events received may not be in the order in which they are generated in external system due to issues like network latency. Therefore, events cannot be processed sequentially based on the order in which they are stored in persistent store. BP Server Event Processing Scheduler (the scheduler) processes the external events. The scheduler processes events, marks events ready for processing, and moves processed events to history. Scheduler processes the event and marks it processed in a single transaction. After that, it periodically moves the events marked as processed into the audit or history table.

The scheduler processes the external events as per the process flow. While processing events, if there are any out-of-sequence external events, then it parks them separately, and processes them later when it reaches their corresponding workstep.
The scheduler periodically checks the parked and yet to be processed events, and selects a batch of eligible (events marked as ready for processing) external events that can be processed. The scheduler selects a batch of external events in the order in which they are created in the external system (that is, sorted by the event creation time), and processes them sequentially.

Lifecycle of a typical external event has the following stages.

1. When the JMS messaging gateway receives an external event, it tries to process the event directly.
2. If the gateway could not process the event, then it persists that event in the external events table.
3. The scheduler picks up the events persisted in the external events table, and processes them.
4. When the process flow reaches a monitoring workstep, BP Server moves the monitoring workstep in Monitoring Wait state, and checks if there are any external events associated with the current workstep, that are to be processed.
5. BP Server processes these external events.
6. If BP Server is configured to store the external events to the history table after processing them, then BP Server marks the processed events as processed in the external events table. A scheduler moves these marked events to the history table in bulk. If BP Server is not configured to store the external events to the history table after processing them, then it deletes these marked events after processing them.

Exception handling

While processing external events, scheduler appropriately and gracefully handles following major exceptions.

- When the event data is invalid.
- When the process state is invalid (process instance / workstep is suspended).
- When Database is not available.
- Any other exceptions generated during the event processing.

Handling duplicate and invalid events

An external event is invalid if it does not contain mandatory data, or contains incorrect data. Scheduler does not process such invalid events. It moves these invalid events directly in the invalid events' table. After a process instance is completed, all the non-processed events for that process instance (identified by its EIId), are moved to the invalid events' table with appropriate reasons.

E-mail notification for invalid events

Along with every event, a list of comma separated e-mail ids can be passed. If the event is found to be invalid, then a notification is sent to all the e-mail ids about the invalid event with proper reasons. In addition to this, a list of e-mail ids can be provided for the smp.invalidevent.notify.email parameter in the OEBPS_HOME\conf\smp.properties file. The notification to these e-mail ids can be controlled by the smp.invalidevent.notify parameter set by default to 'false'.

The different options for e-mail notifications are explained below:

- If the external event contains e-mail ids, then invalid event notification is sent to the provided ids.
- If the external event contains e-mail ids, and in OEBPS_HOME\conf\smp.properties file, the smp.invalidevent.notify parameter is true, then e-mail is sent to all the ids (e-mail ids in the external event, and e-mail ids provided in the smp.properties file for the smp.invalidevent.notify.email parameter).
• If the external event contains e-mail ids, and in OEBPS_HOME\conf\smp.properties file, the smp.invalidevent.notify parameter is false, then e-mail is sent only to the e-mail ids in the external event.

• If the external event contains no e-mail ids, then:
  • In smp.properties file, if e-mail ids are provided for the smp.invalidevent.notify.email parameter and value of the smp.invalidevent.notify parameter is true, then e-mail is sent to all the ids mentioned.
  • In smp.properties file, if e-mail ids are provided for the smp.invalidevent.notify.email parameter and value of the smp.invalidevent.notify parameter is false, then no e-mail is sent.
  • In smp.properties file, if no e-mail ids are provided for the smp.invalidevent.notify.email parameter and value of the smp.invalidevent.notify parameter is true, then no e-mail is sent.

You can customize the default look-and-feel of an invalid event e-mail by modifying the default template file externalevent.html from the OEBPS_HOME\ebmsapps\common\templates\email folder.

**Debugging Monitoring process events**

To enable debugging of Monitoring process, you can set the bpserver.debug.monitoring parameter in the OEBPS_HOME\conf\bpserver.conf file to "true". After setting this parameter, you must refresh the BP Server from BP ServerAdmin using the refreshBL command. Once debugging is enabled, you can see Monitoring process' debug messages in the bpserver.log file.

**Handling out-of-sequence events**

The external activate workstep/completeworkstep events are parked in the persistent store till the target workstep is reached in the process flow. The scheduler periodically checks for the eligible external event, performs the operation specified in the event and then marks the event as processed. Another scheduler job moves these marked events to the history table. While processing the events, the scheduler moves events with incorrect data to invalid events' table.

BP Server can handle the out-of-sequence events if its process instance is not yet created, completed, or removed. Events for completed and removed process instance are moved to invalid events' table. Events received for a process instance before it is created are moved to invalid events' table after making a fixed number of retries.

BP Server can handle following cases of out-of-sequence events.

• Workstep or dataslot update event received before instance is created.
• Event received before or after a workstep is ready.

**Refreshing a Monitoring process**

Like any business process, you can refresh a monitoring process during its life cycle. The following list explains the important points about the monitoring process refresh behavior:

• All the current restrictions and rules of a process refresh apply for monitoring process refresh also.
• You cannot change the dataslot marked as EIId.
• You can add a monitoring workstep, and/or convert a monitoring workstep to a non-monitoring workstep and vice-versa.

• If an existing CHARACTER dataslot is marked as an EIIId dataslot, then the value of this dataslot for all the live process instances must be unique or null, otherwise refresh operation fails.

Using process refresh, you can convert a human workstep containing active work item(s) to a monitoring human workstep. You can then complete the converted monitoring human workstep by an external COMPLETE_WORKSTEP event. However, you cannot complete the work item(s) by an external event. You can complete work item(s), by using APIs or from Business Process Portal.

Migrating a process instance of a monitoring process

You must remember the following rules to migrate process instances of a monitoring process.

• You must map the source EIIId dataslot to a target EIIId dataslot.

• You must migrate a process instance of a monitoring process after all the external events for that process instance are processed, otherwise you must migrate the events manually.

• You must map a monitoring workstep to another monitoring workstep of the same type only.

• After instance migration, if BP Server receives events with old process template name and/or workstep name, then it does not process them.
This chapter explains you about the reporting tables and database views.

For details, see the following topics:

- Reporting tables
- Database views
- Creating views

Reporting tables

BP Server stores the states of active process instances in a set of tables. When a process instance is completed its data is removed from these tables. There is a separate set of tables populated by BPM Process Store which store data about the active as well as completed process instances. You can use these tables for generating reports.

Note: When you uninstall a process template, all data of that process template and all its instances is removed from the reporting tables.

The Table 33: Reporting tables on page 256 lists the important reporting tables.
Table 33: Reporting tables

<table>
<thead>
<tr>
<th>Table name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROCESSTEMPLE</td>
<td>This table stores data of install process templates.</td>
</tr>
<tr>
<td>PROCESSDATAINFO</td>
<td>This table stores metadata of dataslots of all installed process templates.</td>
</tr>
<tr>
<td>PROCESSWORKSTEPINFO</td>
<td>This table stores metadata of worksteps of all installed process templates.</td>
</tr>
<tr>
<td>PROCESSINSTANCE</td>
<td>This table stores data of active and completed process instances.</td>
</tr>
<tr>
<td>WORKSTEP</td>
<td>This table stores data of active and completed workstep instances.</td>
</tr>
<tr>
<td>WORKITEM</td>
<td>This table stores data of active and completed workitems.</td>
</tr>
<tr>
<td>SUBPROCESSINFO</td>
<td>This table stores data of all process instances and their parent processes.</td>
</tr>
<tr>
<td>CWORKITEM</td>
<td>This table stores data of collaborative workstep items.</td>
</tr>
<tr>
<td>CSTEP</td>
<td>This table stores data of collaborative steps.</td>
</tr>
<tr>
<td>PROCESS_NOTES</td>
<td>This table stores data of all process notes</td>
</tr>
</tbody>
</table>

Note: You can use the archiver to purge the data from these reporting tables. For more information, refer to the Server Administrator's Guide.

Database views

You can use the database views while designing Online Analytical Processing (OLAP) reports using Progress Analyst Studio. These database views are BPM Process Store tables that store application data. Business Process Server provides the following database views.

**PROCESS_INSTANCE_VIEW**

This view helps you to analyze process data on various process parameters like duration, count, and status. Attribute names and description of each column in this table are given in Table 34: PROCESS_INSTANCE_VIEW table on page 256.

Table 34: PROCESS_INSTANCE_VIEW table

<table>
<thead>
<tr>
<th>Attribute name</th>
<th>Column name</th>
<th>Table name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProcessInstanceId</td>
<td>PROCESS_INSTANCE_ID</td>
<td>PROCESS_INSTANCE</td>
<td>Process instance ID.</td>
</tr>
<tr>
<td>ProcessName</td>
<td>PROCESS_TEMPLATE_NAME</td>
<td>PROCESS_TEMPLATE</td>
<td>OpenEdge application template name.</td>
</tr>
<tr>
<td>ProcessStatus</td>
<td>STATUS</td>
<td>PROCESS_INSTANCE</td>
<td>Status of the process instance.</td>
</tr>
<tr>
<td>Attribute name</td>
<td>Column name</td>
<td>Table name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>----------------</td>
<td>------------</td>
<td>---------------------------------------</td>
</tr>
<tr>
<td>ProcessPriority</td>
<td>PRIORITY</td>
<td>PROCESS INSTANCE</td>
<td>Priority of the process instance.</td>
</tr>
<tr>
<td>Start</td>
<td>START_TIME</td>
<td>PROCESS INSTANCE</td>
<td>Start time of the process instance.</td>
</tr>
<tr>
<td>End</td>
<td>END_TIME</td>
<td>PROCESS INSTANCE</td>
<td>End time of the process instance.</td>
</tr>
<tr>
<td>DueDate</td>
<td>DUE_DATE</td>
<td>PROCESS INSTANCE</td>
<td>Due date of the process instance.</td>
</tr>
<tr>
<td>ActualDuration</td>
<td>DURATION</td>
<td>PROCESS INSTANCE</td>
<td>Actual duration of the process instance.</td>
</tr>
<tr>
<td>EstimatedDuration</td>
<td>ESTIMATED_DURATION</td>
<td>PROCESS INSTANCE</td>
<td>Estimated duration of the process instance.</td>
</tr>
<tr>
<td>Creator</td>
<td>CREATOR</td>
<td>PROCESS INSTANCE</td>
<td>Creator of the process instance.</td>
</tr>
</tbody>
</table>

**PROCESS_WORKSTEP_VIEW**

This view helps you to analyze process data on various workstep parameters like duration, count, and status. Attribute names and description of each column in this table is given in Table 35: PROCESS_WORKSTEP_VIEW table on page 257.

This view is filtered on the following conditions.

- **Type of workstep must be** ATOMIC or NESTED, for Monitoring workstep it must be EXTERNAL. All other types of worksteps are filtered.

- **Workstep status must not be** W_EVENTACTIVATION_WAIT.

**Table 35: PROCESS_WORKSTEP_VIEW table**

<table>
<thead>
<tr>
<th>Attribute name</th>
<th>Column name</th>
<th>Table name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProcessInstanceId</td>
<td>PROCESS_INSTANCE_ID</td>
<td>WORKSTEP</td>
<td>Process instance ID.</td>
</tr>
<tr>
<td>ProcessName</td>
<td>PROCESS_TEMPLATE_NAME</td>
<td>PROCESS TEMPLATE</td>
<td>OpenEdge application template name.</td>
</tr>
<tr>
<td>ProcessStatus</td>
<td>STATUS</td>
<td>PROCESS INSTANCE</td>
<td>Status of the process instance.</td>
</tr>
<tr>
<td>ProcessPriority</td>
<td>PRIORITY</td>
<td>PROCESS INSTANCE</td>
<td>Priority of the process instance.</td>
</tr>
<tr>
<td>WorkstepName</td>
<td>WORKSTEP_NAME</td>
<td>WORKSTEP</td>
<td>Workstep name.</td>
</tr>
<tr>
<td>Attribute name</td>
<td>Column name</td>
<td>Table name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------</td>
<td>------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>WorkstepPriority</td>
<td>PRIORITY</td>
<td>WORKSTEP</td>
<td>Priority of the workstep.</td>
</tr>
<tr>
<td>WorkstepStatus</td>
<td>STATUS</td>
<td>WORKSTEP</td>
<td>Status of the workstep.</td>
</tr>
<tr>
<td>LoopCounter</td>
<td>LOOPCOUNTER</td>
<td>WORKSTEP</td>
<td>Loop counter of workstep.</td>
</tr>
<tr>
<td>ProcessTemplateId</td>
<td>PROCESS_TEMPLATE_ID</td>
<td>WORKSTEP</td>
<td>OpenEdge application template ID.</td>
</tr>
<tr>
<td>Start</td>
<td>START_TIME</td>
<td>WORKSTEP</td>
<td>Start time of the workstep.</td>
</tr>
<tr>
<td>End</td>
<td>END_TIME</td>
<td>WORKSTEP</td>
<td>End time of workstep.</td>
</tr>
<tr>
<td>ActualDuration</td>
<td>DURATION</td>
<td>WORKSTEP</td>
<td>Actual duration of the workstep.</td>
</tr>
<tr>
<td>EstimatedDuration</td>
<td>ESTIMATED_DURATION</td>
<td>WORKSTEP</td>
<td>Estimated duration of the workstep.</td>
</tr>
<tr>
<td>DueDate</td>
<td>DUE_DATE</td>
<td>WORKSTEP</td>
<td>Due date of the workstep.</td>
</tr>
<tr>
<td>Type</td>
<td>TYPE</td>
<td>WORKSTEP</td>
<td>Workstep type.</td>
</tr>
<tr>
<td>IsMonitorStep</td>
<td>MONITOR_STEP</td>
<td>WORKSTEP</td>
<td>Whether it is a monitoring workstep. Value ‘1’ indicates it is a monitoring workstep.</td>
</tr>
<tr>
<td>Creator</td>
<td>CREATOR</td>
<td>PROCESS_INSTANCE</td>
<td>Creator of the process instance.</td>
</tr>
<tr>
<td>Performer</td>
<td>PERFORMER</td>
<td>WORKSTEP</td>
<td>Workstep performer.</td>
</tr>
</tbody>
</table>

**PROCESS_WORKITEM_VIEW**

This view helps you to analyze process data on the work item duration, count, status, and performer load. Attribute names and description of each column in this table are given in Table 36: PROCESS_WORKITEM_VIEW table on page 258.

Table 36: PROCESS_WORKITEM_VIEW table

<table>
<thead>
<tr>
<th>Attribute name</th>
<th>Column name</th>
<th>Table name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WorkItemId</td>
<td>WORKITEM_ID</td>
<td>WORKITEM</td>
<td>Work item ID.</td>
</tr>
<tr>
<td>ProcessTemplateId</td>
<td>PROCESS_TEMPLATE_ID</td>
<td>WORKITEM</td>
<td>OpenEdge application template ID.</td>
</tr>
<tr>
<td>ProcessName</td>
<td>PROCESS_TEMPLATE_NAME</td>
<td>PROCESS_TEMPLATE</td>
<td>OpenEdge application template name.</td>
</tr>
<tr>
<td>ProcessInstanceId</td>
<td>PROCESS_INSTANCE_ID</td>
<td>WORKITEM</td>
<td>Process instance ID.</td>
</tr>
</tbody>
</table>
**Dynamic process view**

For every installed process, Business Process Server creates a non-materialized view with name `<PROCESSNAME>_VIEW` in the database. This view helps you to analyze process data associated with instances of that process. Attribute names and description of each column in this view are given in Table 37: Dynamic process view on page 259.

**Table 37: Dynamic process view**

<table>
<thead>
<tr>
<th>Attribute name</th>
<th>Column name</th>
<th>Table name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProcessInstanceId</td>
<td>PROCESS_INSTANCE_ID</td>
<td>PROCESS_INSTANCE</td>
<td>Process instance ID.</td>
</tr>
<tr>
<td>ProcessName</td>
<td>PROCESS_TEMPLATE_NAME</td>
<td>PROCESS_TEMPLATE</td>
<td>OpenEdge application template name.</td>
</tr>
<tr>
<td>ProcessStatus</td>
<td>PROCESS_STATUS</td>
<td>PROCESS_INSTANCE</td>
<td>Status of the process instance.</td>
</tr>
<tr>
<td>ProcessPriority</td>
<td>PROCESS_PRIORITY</td>
<td>PROCESS_INSTANCE</td>
<td>Priority of the process instance.</td>
</tr>
<tr>
<td>Start</td>
<td>START_TIME</td>
<td>PROCESS_INSTANCE</td>
<td>Start time of the process instance.</td>
</tr>
<tr>
<td>End</td>
<td>END_TIME</td>
<td>PROCESS_INSTANCE</td>
<td>End time of the process instance.</td>
</tr>
</tbody>
</table>
If the process is a monitoring process, then its External instance ID (EIID) value is stored in column 'EIID' which is the first column after column 'CREATOR'.

For example, Business Process Server creates the dynamic process view for the sample application 'AccountsReceivable' with the name 'ACCOUNTSRECEIVABLE_VIEW'. Attribute names and description of each column in this view are given in Table 38: ACCOUNTSRECEIVABLE_VIEW on page 260.

### Table 38: ACCOUNTSRECEIVABLE_VIEW

<table>
<thead>
<tr>
<th>Attribute name</th>
<th>Column name</th>
<th>Table name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProcessInstanceId</td>
<td>PROCESS_INSTANCE_ID</td>
<td>PROCESS_INSTANCE</td>
<td>Process instance ID.</td>
</tr>
<tr>
<td>ProcessName</td>
<td>PROCESS_TEMPLATE_NAME</td>
<td>PROCESS_TEMPLATE</td>
<td>OpenEdge application template name.</td>
</tr>
<tr>
<td>ProcessStatus</td>
<td>PROCESS_STATUS</td>
<td>PROCESS_INSTANCE</td>
<td>Status of the process instance.</td>
</tr>
<tr>
<td>ProcessPriority</td>
<td>PROCESS_PRIORITY</td>
<td>PROCESS_INSTANCE</td>
<td>Priority of the process instance.</td>
</tr>
<tr>
<td>Start</td>
<td>START_TIME</td>
<td>PROCESS_INSTANCE</td>
<td>Start time of the process instance.</td>
</tr>
<tr>
<td>End</td>
<td>END_TIME</td>
<td>PROCESS_INSTANCE</td>
<td>End time of the process instance.</td>
</tr>
<tr>
<td>Duration</td>
<td>DURATION</td>
<td>PROCESS_INSTANCE</td>
<td>Actual duration of the process instance.</td>
</tr>
<tr>
<td>Creator</td>
<td>CREATOR</td>
<td>PROCESS_INSTANCE</td>
<td>Creator of the process instance.</td>
</tr>
</tbody>
</table>
Creating views

To create database views:

- Run the following command from the OEBPS_HOME/bin folder

  setupoebps -c ProcessViews

  This creates database views that can be used while designing OLAP reports using Progress Analyst Studio.
**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.