Administering webMethods Process Engine

Version 10.5

October 2019

WEBMETHODS

This document applies to webMethods Process Engine Version 10.5 and to all subsequent releases.

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

Document ID: PE-AG-105-20191015

Table of Contents

About this Guide

This guide describes how the Process Engine controls the run-time execution of business processes designed in Software AG Designer. It explains how to tune Process Engines to achieve the right balance of quality of service and performance, and how to configure Process Engines to work together in a distributed environment.

Additional information about processes can be found in the Software AG Designer online help, *Monitoring BPM, Services, and Documents with BAM: webMethods Monitor User's Guide*, and the *Getting Started with BPM* guide. This guide assumes you are already familiar with process model design and business process monitoring.

Deprecation of webMethods Broker

webMethods Broker is deprecated for use with webMethods 10.2. If you are starting development using webMethods 10.2, you should use webMethods Universal Messaging instead of webMethods Broker. If you are upgrading to webMethods 10.2, you should consider migrating to Universal Messaging. If you choose to continue to use webMethods Broker, you will still be fully supported, but only until the announced end-of-life dates for webMethods Broker. For details, see "https://empower.softwareag.com/brokerendoflife/"

Document Conventions

Convention	Description
Bold	Identifies elements on a screen.
Narrowfont	Identifies service names and locations in the format *folder.subfolder.service* , APIs, Java classes, methods, properties.
Italic	Identifies: Variables for which you must supply values specific to your own situation or environment. New terms the first time they occur in the text. References to other documentation sources.
`Monospace font`	Identifies: Text you must type in.

Convention	Description
	Messages displayed by the system. Program code.
{ }	Indicates a set of choices from which you must choose one. Type only the information inside the curly braces. Do not type the { } symbols.
|	Separates two mutually exclusive choices in a syntax line. Type one of these choices. Do not type the | symbol.
[ ]	Indicates one or more options. Type only the information inside the square brackets. Do not type the [ ] symbols.
...	Indicates that you can type multiple options of the same type. Type only the information. Do not type the ellipsis (...).

Online Information and Support

Software AG Documentation Website

You can find documentation on the Software AG Documentation website at "http://documentation.softwareag.com". The site requires credentials for Software AG's Product Support site Empower. If you do not have Empower credentials, you must use the TECHcommunity website.

Software AG Empower Product Support Website

If you do not yet have an account for Empower, send an email to "empower@softwareag.com" with your name, company, and company email address and request an account.

Once you have an account, you can open Support Incidents online via the eService section of Empower at "https://empower.softwareag.com/".

You can find product information on the Software AG Empower Product Support website at "https://empower.softwareag.com".

To submit feature/enhancement requests, get information about product availability, and download products, go to "Products".

To get information about fixes and to read early warnings, technical papers, and knowledge base articles, go to the "Knowledge Center".

If you have any questions, you can find a local or toll-free number for your country in our Global Support Contact Directory at "https://empower.softwareag.com/public_directory.asp" and give us a call.

Software AG TECHcommunity

You can find documentation and other technical information on the Software AG TECHcommunity website at "http://techcommunity.softwareag.com". You can:

- Access product documentation, if you have TECHcommunity credentials. If you do not, you will need to register and specify "Documentation" as an area of interest.
- Access articles, code samples, demos, and tutorials.
- Use the online discussion forums, moderated by Software AG professionals, to ask questions, discuss best practices, and learn how other customers are using Software AG technology.
- Link to external websites that discuss open standards and web technology.

Data Protection

Software AG products provide functionality with respect to processing of personal data according to the EU General Data Protection Regulation (GDPR). Where applicable, appropriate steps are documented in the respective administration documentation.

1 Concepts

About Process Engine

Process Engine controls the run-time execution of business processes. Process Engine is a webMethods Integration Server package (WmPRT) that you install on every Integration Server that is used to run steps in a business process model designed in Software AG Designer.

In a production environment, multiple Integration Servers are typically running in a clustered environment. For more information about the behavior of Process Engine in a clustered environment, see "About the Process Engine and Clustered Operation" on page 22.

Tuning Process Engine Performance and Quality of Service

You can tune a process model for quality of service and performance. Quality of service is a measure of reliability, visibility, and control. In general, you achieve high quality of service by persisting data as a process executes. Performance is a measure of the time it takes a process to complete and the number of processes completed per time period. In general, you achieve good performance by using volatile data storage (server RAM) while a process executes. An increased quality of service is achieved at the expense of performance and vice versa.

In a maximum quality of service/minimum performance environment, processes are guaranteed; that is, a process automatically recovers at the appropriate step in case of system failure, and the process is guaranteed to run to completion. However, in this case, each process takes longer to complete and fewer instances complete per time period.

In a maximum performance/minimum quality of service environment, processes complete more quickly and more processes complete per time period. However, processes are not guaranteed and might not automatically recover in case of server failure; that is, if a server fails, the process might not run to completion. In addition, there is no visibility or control through webMethods Monitor. You cannot take any of the actions through webMethods Monitor that are available in a maximum quality of service environment.

The level of quality of service and performance that you require depends on your business needs. The Process Engine quality of service and performance settings are highly configurable and can be set on a process model by process model basis; you can tune your environment to a quality of service or performance extreme or anywhere in between.

Storing Process Status Information

Process status information can be stored in the Process Engine database component (the default configuration), or it can be stored in RAM. You select the method of storage by setting the **Volatile Tracking** option in the process properties of Designer (different process models may use different configuration settings). A Process Engine stores the details of a process—such as start time, end time, and whether is it active or disable—in the selected location.

For more information about setting the **Volatile Tracking** option, see "Setting Quality of Service for a Process" on page 26. For more information about how the Process Engine maintains status data, see the topic "Tracking Process Status" in the *webMethods BPM Process Development Help*.

Logging Process Data

When you design a process model, you specify the amount and type of data to log for its processes. As the processes run, the Process Engine can log the process status. You can then monitor and control processes by accessing the process logging data through webMethods Monitor.

For instructions on setting up process logging, see the PDF publication *webMethods Monitor User's Guide*.

Working with Built-in Services

The Process Engine offers predefined (or *built-in*) services that enable you to manage processes and perform a wide range of actions within processes, such as creating cross-references, creating a custom process instance ID, or performing custom logging.

For information about the available Process Engine built-in services, see "Process Engine Services" on page 53. Process-oriented built-in services are also provided with webMethods Monitor; for more information, see the *webMethods Monitor Built-In Services Reference*.

Viewing and Controlling Processes

webMethods Monitor provides you with maximum visibility and control of processes. You can use webMethods Monitor to:

- View the status of processes and steps.
- Suspend, resume, resubmit, and stop processes.

- View error and activity messages.
- Edit and resubmit processes at specific steps.

The webMethods Monitor user interface is available in My webMethods (the webMethods Monitor interface component must be installed in My webMethods Server, either at the time of initial installation, or afterward). For more information about working with webMethods Monitor, see the PDF publication *webMethods Monitor User's Guide*. For more information about working with My webMethods, see the PDF publication *Working with My webMethods*.

About the Process Engine Storage Cleaner

The Process Engine contains an internal utility known as the Storage Cleaner. The Storage Cleaner runs periodically and removes older process instance data from the Process Engine database, thus preventing the database from growing too large.

Storage Cleaner operation is defined by these three Process Engine settings:

- **Cleanup Service Execution Interval**. Default = 600 seconds. The interval at which Storage Cleaner runs, deleting any Process Engine table data that is eligible for deletion, as defined by the following two settings:
- **Completed Process Expiration**. Default = 60 seconds. The period of time that must elapse before Completed process instance data becomes eligible for deletion.
- **Failed Process Expiration**. Default = 600 seconds. The period of time that must elapse before Failed process instance data becomes eligible for deletion.

For information about modifying these settings, see "Configuring Process Engine Settings" on page 37. The operation of the Storage Cleaner can be disabled completely. For more information, see "Disabling/Enabling the Process Engine Storage Cleaner" on page 41.

Note: A separate Process Engine setting, **Cache Cleanup Interval**, determines how often the Process Engine cleans up certain tables held in memory. This mechanism is totally separate from the Storage Cleaner.

Special Considerations in Clustered Operation

In an environment where two or more Process Engines are operated in a cluster, you must ensure that Storage Cleaner is enabled on at least one node in the cluster, and preferably on multiple nodes.

This is especially true if the cluster configuration is not stable over time, that is, nodes are often taken offline and then brought back online.

Important: If a cluster node that has Storage Cleaner enabled is taken offline or removed from the cluster, be sure that Storage Cleaner is enabled on at least one other

node in the cluster. If you do not, Process Engine database size limits may be reached, and problems may be experienced if you add the offline node back into the cluster. The previously offline node will attempt to sync to the Process Engine database, and if no Storage Cleaner has been running, the data transfer is likely to be large and out-of-memory errors may result. When Storage Cleaner eventually does run, it is likely to take a long time to complete.

In situations where cluster operation is reliably stable, you can enable Storage Cleaner on multiple nodes in a cluster to share load. Process Engine uses a lock-management system to serialize the operation of Storage Cleaners on multiple nodes, so no two nodes will be running the Storage Cleaner at the same time.

2 Process Engine-Related Run-Time Items

Overview

When you generate a process model, Designer creates an Integration Server package on each of the Integration Servers that are defined as the logical servers used within the process model steps. The package is stored in the *Integration Server_directory*\instances *instance_name* \packages directory under the name *package_name* , where *package_name* is the project name or the user-defined package name that is generated from the name you specified in Designer.

Inside the package, Designer generates several items required for the execution of the process, including:

- Triggers, which are used to start a process based on a received document or message.
- Services representing each step in the process model.
- Process description files.

This chapter describes how these items relate to process execution, process tracking, and the Process Engine.

Note: The Software AG Designer online help contains additional information about triggers and JMS connection aliases.

Generated Process Model Triggers

Process Engine works with two different types of triggers:

- webMethods Broker (deprecated) triggers. These triggers require publishing of documents using the Integration Server publish/subscribe services or the webMethods Broker (deprecated) APIs.
- JMS triggers. JMS triggers allow third-party JMS clients to send messages to trigger processes.
- Software AG Universal Messaging triggers. The Universal Messaging triggers support both the JMS and the publish/subscribe protocols.

In general, triggers work automatically within the process execution framework. To view the status of the available process triggers, log in to Integration Server Administrator and browse to **Settings > Messaging** Both webMethods Broker (deprecated) and JMS triggers are available for viewing and editing on the Trigger Management pages.

Trigger Details

When a process model version is generated, the Integration Server package created by Designer contains the following triggers for each logical server to which steps are assigned in the process model version:

- Subscription triggers. Designer uses these to manage external input document subscriptions for *all* versions of the process model. There is exactly one subscription trigger for each version of the process model for each logical server defined to the process model.

  Note that the subscription document structure must be the same for each version of the process model (unless the model was generated to use JMS triggers). If you change the subscription document for a new process model version, your subscriptions will fail to work correctly for older versions of the process model (unless JMS triggers are used).

  Subscription triggers are also used to define retry behavior for intermediate receive steps; for more information, see the topic "Correlation Behavior with Non-starting Receive Tasks" in the *webMethods BPM Process Development Help*.

- Transition triggers. Designer generates a transition trigger for each version of the process model to process internal transition documents. There is exactly one transition trigger for each version of the process model for each logical server defined to the process model. Each *process transition document* contains the process model ID, the process model version number, the instance ID, the step ID and outputs from the step that produced the document, the step ID for the next step to run, and the source server ID.

Note: JMS trigger generation is available for subscription and transition triggers. In earlier releases, both of these triggers used the same JMS connection alias, PE_NONTRANSACTIONAL_ALIAS. With version 9.0 and later, you can pick a connection alias of your own choosing for message events. This will result in the subscription trigger using the selected alias, but the transition trigger will still use the PE_NONTRANSACTIONAL_ALIAS alias. For additional information, see "About Process Engine and JMS Connection Aliases", below.

Process Engine also uses the following additional triggers:

- Broadcast trigger. This trigger handles broadcast events to other nodes.
- Restart trigger. This trigger is used only when webMethods Monitor generates a resubmit request.

These two triggers require you create two specific JMS topics, as described in "Configuring Process Engine for Universal Messaging " on page 47.

About Subscription Protocol Connection Aliases

When you build and upload a subscription triggered process, Process Engine uses the default Integration Server connection alias, based on the message provider configured in Integration Server. The default connection alias for webMethods Broker (deprecated) is IS_BROKER_CONNECTION. The default connection alias for Universal Messaging is IS_UM_CONNECTION. When you change the default connection alias, you must reload the WmPRT package in Integration Server Administrator.

When you create a custom IS connection alias, you must set it as the default connection alias. Process Engine always uses the default IS connection alias.

About Process Engine and JMS Connection Aliases

By default, the Process Engine uses a non-transactional JMS connection alias with the specific name PE_NONTRANSACTIONAL_ALIAS when either of the following conditions is true:

- You are running the Process Engine in a multi-Integration Server environment (that is, either in a cluster or using distributed models).

  OR

- You are running a single Integration Server configuration and running process models generated with receive steps that use the JMS protocol.

Note: The PE_NONTRANSACTIONAL_ALIAS is created automatically during installation with a default value of `DEFAULT_IS_JNDI_PROVIDER`. If only a webMethods Broker (deprecated) instance is found on the local server, then `DEFAULT_IS_JNDI_PROVIDER` is set to point to the webMethods Broker (deprecated) server. If both webMethods Broker (deprecated) and Software AG Universal Messaging are installed, or Universal Messaging only is installed, then `DEFAULT_IS_JNDI_PROVIDER` is set to point to the Universal Messaging server. If for any reason you create this alias manually, you must reload the WmPRT package after you complete the alias creation.

In the above cases, JMS *must be configured* on your Integration Server(s) as follows:

- In a clustered or distributed Integration Server environment, the Connection Client ID (JMS) *must be identical* for all nodes.
- In a non-clustered environment, the Connection Client ID (JMS) *must be different* for each node.

A JMS connection alias is required because the Process Engine uses it to broadcast internal events between the Process Engine nodes. When you select the JMS protocol, Process Engine uses the PE_NONTRANSACTIONAL_ALIAS connection alias by default for communication, and generates a destination in the subscription trigger of the generated process. When you publish a message using JMS and the

PE_NONTRANSACTIONAL_ALIAS to trigger the process, you must specify this auto-generated destination. For convenience, the pub.prt.jms.send service can be used to simplify sending messages that trigger processes.

In addition to the default connection alias, you can create and specify custom JMS aliases for use with subscription triggers. You must create these connection aliases manually on Integration Server. Custom JMS connection alias names are stored in the process, and custom alias definitions are stored in Integration Server and on webMethods Broker (deprecated).

- When you use a custom JMS alias, you can also use a custom destination in the subscription trigger of the generated process. Custom destinations must also be created manually. When you publish a message using JMS and a custom connection alias, you must specify the destination.
- When you receive a message using JMS and a custom connection alias, you can specify a destination, or you can use the default destination created by Designer.
- Transition triggers will always use the PE_NONTRANSACTIONAL_ALIAS connection alias.

To learn more about creating JMS connection aliases and working with JMS triggers, see the PDF publication *Using webMethods Integration Server to Build a Client for JMS*.

Note: When a JMS-triggered process uses the PE_NONTRANSACTIONAL_ALIAS connection alias, that alias must have the **Manage Destinations** option shown as **Yes** in the connection alias Advanced Settings. For more information, see "Setting the Manage Destination Option for a Connection Alias" in the *webMethods BPM Process Development Help*.

About Parallel Execution (Step Locking)

The Process Engine supports the concept of the *parallel execution* of steps, also known as *step locking*. You implement step locking on a per-step basis, by selecting the **Allow parallel execution** check box in the **Implementation** tab of the step properties in Designer to specify that the step must be locked at run time. Locking a step allows it to be executed by multiple threads.

In Process Engine 7.1.1 and earlier versions, step locking was always enabled for receive and join steps, but disabled for all other step types. In Process Engine 7.1.2 and later versions, step locking is available as an option that can be enabled or disabled for each individual step.

If you plan to include parallel step execution in your process model (including referenced process steps), you must enable those steps for step locking by selecting the **Allow parallel execution** check box in the step's Properties view in Software AG Designer. Step locking is always recommended for use in re-entrant processes, especially during heavy processing loads. A re-entrant process is a process with multiple documents with the same correlation joining to the same instance in different paths. If step locking is not implemented, a receive step can intermittently display the waiting state.

Process Description Files

The Integration Server package that Designer generates on each server also contains a *process description file* for that server. These files have a file name extension of .frag, and are sometimes referred to as frag files. The process description file for each server contains the following:

- Process model version property settings.
- For each step that will run on the server, the step definition and the step ID for the next step to run.
- If the process model version uses an express pipeline to pass data from step to step, a list of all inputs explicitly specified for downstream steps in the model version.
- If the process model version uses correlation IDs, the correlation service name and properties.

You should have no need to edit a process description file.

About the Process Engine and Clustered Operation

In its most basic implementation, the Process Engine is installed as the WmPRT package on a single Integration Server, connected to its own Process Engine database. A webMethods Broker (deprecated) is also included for routing and delivering messages.

For process model development and testing purposes, or for very small process solutions, a single Process Engine, Integration Server, and webMethods Broker (deprecated) is adequate. However, most process solutions require more resources for efficient operation, and it is common practice to run two or more Integration Servers in one of the following configurations:

- Integration Server cluster. All Integration Server nodes share the same webMethods Broker (deprecated) client connection so that only one node processes a message. For details, see "Operating Process Engine in an Integration Server Cluster" on page 23.
- Distributed (non-clustered) mode. All Integration Server nodes have their own webMethods Broker (deprecated) client connection but the process model is designed so that only specific parts of the process run on any one Integration Server node. For details, see "Operating Process Engine in an Integration Server Cluster" on page 23.
- External cluster. This is functionally equivalent to the Integration Server cluster from a BPM perspective, except that the Integration Server nodes themselves are not part of a Integration Server cluster. Customers can chose this configuration when they do not want to use the actual Integration Server clustering feature. For details, see "Operating Process Engine in an Integration Server Cluster" on page 23.

For more information about working with Integration Server and webMethods Broker (deprecated) in a clustered environment, see *webMethods Integration Server Clustering Guide*.

Operating Process Engine in an Integration Server Cluster

When two or more Integration Servers are running in a clustered environment, the Process Engines within them are running in a clustered environment as well.

Note: Although the term "Process Engine cluster" may be used from time to time, it actually refers to Process Engines running in an Integration Server cluster.

When you run a process in a clustered Integration Server environment, all Integration Servers that run steps in that process are connected to webMethods Broker (deprecated) using the same client prefix. webMethods Broker (deprecated) serves as the communication link for all of the Process Engines in the Integration Server cluster.

Within the cluster, all steps of a generated and uploaded process model exist on all Integration Server nodes. In other words, from a process point of view, each node in the cluster is an exact copy of the other nodes. The shared webMethods Broker (deprecated) ensures that each process-related message is routed to one and only one Integration Server in the cluster, so there is no duplicate processing.

When two or more Process Engines are running in an Integration Server cluster, all of the Process Engines share a single Process Engine database. Software AG does not provide support for Process Engines to access or share more than one database. See the PDF publication *webMethods Integration Server Clustering Guide* for information about setting up the external RDBMS.

Operating Process Engine in Distributed Cluster Mode

There may be situations where you want to distribute your processing over multiple Process Engines, but distribute the steps of a process model among the Integration Server nodes in a cluster.

In this implementation, you design the process model so that generation distributes the steps among the Integration Server nodes. That is, the sum total of all steps across all Integration Server nodes in a distributed cluster equals the complete process. A distributed cluster implementation shares a webMethods Broker (deprecated), but each Integration Server connects to the central webMethods Broker (deprecated) instance with its own client prefix.

As in a standard cluster, a single external RDBMS containing a Process Engine database supports all Integration Server nodes in a distributed implementation.

Operating Process Engine in an External Cluster

An external cluster enables you to distribute your processes among multiple Integration Servers without fully implementing an Integration Server cluster. In this case, although the Integration Server nodes are not truly clustered, they must still connect to webMethods Broker (deprecated) using the same client prefix.

Because the Integration Server nodes are not truly clustered, the Process Engine on each node must be configured to behave as if it was operating in a clustered environment. This ensures that each Process Engine executes processes in compliance with cluster requirements (locking step context, for example).

This is done by enabling the External Cluster setting for each Process Engine, as described in "Configuring Process Engine Settings" on page 37. When this setting is enabled, the Process Engine behaves as if it was in a Integration Server cluster.

Each Integration Server must communicate with the webMethods Broker (deprecated) using the same client prefix. As in a standard cluster, you must configure each Integration Server node to use a single external RDBMS containing a Process Engine database.

Upgrading Process Model Versions

In Software AG Designer, you can modify an existing process model and create a new version of the model. You can then build and upload the new version, and then enable it for use. In doing so, you can update any or all of the currently running instances of that model so that they start using the newer version. For more information, see these topics:

- "About Process Versions" in the *webMethods BPM Process Development Help*.
- "Updating a Process Instance to a New Model Version" in the PDF publication *webMethods Monitor User's Guide*.
- "Enabling and Disabling Process Model Versions" in the PDF publication *webMethods Monitor User's Guide*.

3 Tuning Process Engine Performance and Quality of Service

Overview of Tuning

You can optimize, or *tune*, Process Engine performance and quality of service for processes by editing process model properties. These properties apply to all Process Engines on Integration Servers that will run steps in the process model.

You set process properties in the following locations:

- In Software AG Designer (at design time); these settings include:
  - Local optimization
  - Express pipeline
  - Volatile transition documents
  - Volatile tracking
  - Minimum logging level
- In webMethods Monitor (at run time); these settings include:
  - Transition logging
  - Logging level
  - Enabling resubmissions.

Note: For processes that reference other processes, the Process Engine uses the process property settings for the process, and the referenced process property settings for the referenced process.

Setting Quality of Service for a Process

You can define quality of service settings when you design a process. These settings are made in Software AG Designer and are used to determine how a process executes at run time, enabling you to select a balance of performance, reliability, visibility, and control.

To specify the quality of service settings for a process in Designer

1. On the Process Development perspective, open the process you want to work with.
2. Click the Properties view if it is not already visible.
3. On the **Advanced** tab, click **Run Time**.
4. Using the following tables, enter your selections for the quality of service settings.

Quality of Service Setting	Description
Optimize Locally	Execute adjacent steps on the same Integration Server without publishing transition documents. Enabled by default. ■ Select **Optimize Locally** when you want to use a pipeline to pass data from step to step on the local server, and publish a process transition document *only* when there is a transition to a step running on another server, or if a process splits into more than one branch. ■ Clear this check box to always publish a process transition document when transitioning to any step, no matter where it is located. No pipeline is used. You can select **Optimize Locally** to decrease document message traffic and improve performance. However, if a step fails, the process can recover automatically only at the step that published the most recent process transition document, and that step might not be the step of failure. For example: Suppose process step 1 runs on Server A and process steps 2, 3, and 4 run on Server B. When you are optimizing locally and step 3 fails, the most recently published process transition document was produced by step 1, because the Process Engine did not publish a process transition document for step 2 or 3. The process, therefore, recovers automatically at step 1 (that is, step 2 will be run again). When you do not optimize locally, every step publishes a process transition document, so a process can automatically recover at the step it completed last. In this case, the process recovers at step 3. The biggest risk of optimizing locally is duplication of work. For example, you might not want to risk duplicating work for processes that store, synchronize, or correlate data. For processes that do less critical work, the performance benefits might outweigh the risks. When a referenced process is invoked and **Optimize Locally** is selected, the Process Engine will attempt to locally invoke a referenced process, with the following conditions applied: ■ The referenced process exists on the same Process Engine node as the parent step. ■ The referenced process has no subscription filter. ■ If the Integration Server thread usage is below the threshold, communication is done with a direct service invocation on a new Integration Server thread. Otherwise, communication is through

Quality of Service Setting	Description
	the publishing of a document that is handled on the appropriate model trigger. **Note:** The above thread behavior applies to all types of child invocations, including static and dynamic reference processes, and static and dynamic callable processes. Subscription filters are enforced at the trigger level. If there is a filter on a referenced process, the Process Engine will ignore the **Optimize Locally** setting and publish the transition document. When **Optimize Locally** is selected and data is returned from the referenced process to the parent, the parent step must be running on the same Process Engine node as the referenced process for successful data transfer.
Express Pipeline	Send a reduced data set between the steps in a process. Enabled by default. Using the express pipeline can significantly improve performance when pipelines are large (1 MB or more). This setting applies to both the pipeline and transition document methods of transferring data and is independent of the **Optimize Locally** setting. ■ Select **Express Pipeline** to specify that you want to pass a reduced (*express*) data set from step to step. ■ Clear this check box to specify that the complete data set is passed from step to step. **Note:** When a process is resubmitted, the complete data set is passed, regardless of this setting. When you use the complete data set, the server passes all data from step to step, regardless of whether outputs are used by downstream steps. With the Express Pipeline option enabled, the server reads the list of inputs in the process description file and passes a reduced data set that contains *only those outputs explicitly specified in the process model version as inputs* to following steps. All other data is discarded. Do *not* use the Express Pipeline setting for:

- Processes that include steps with services that add values to the pipeline data; the server will not include the added values because they are not explicitly specified as inputs to downstream steps.
- Processes that contain a process-wide error handler step; the process-wide error handler step is not recognized as a downstream step and will therefore not receive the necessary input data.

Important: Designer *never* implements Express Pipeline for dynamic referenced process/call activity steps, regardless of the **Express Pipeline** option setting.

The purpose of Express Pipeline is to protect explicitly-defined pipe elements, such as step inputs and outputs, from being removed at run-time. Early in your development cycle, however, you may not yet have added any inputs/outputs to the steps in your model, and thus your model is incomplete and not thoroughly designed. In such cases, when you generate (build and upload) your model, then the Express Pipeline setting displayed on the WmPRT home page will not match the value of the Express Pipeline check box as set in Designer.

Important: When the design is not yet complete, that is you have not yet added any inputs/outputs to the steps, then Express Pipeline in the WmPRT home page will always be displayed as "No". However, during runtime Process Engine will always correctly use the Express Pipeline value as set in Designer.

Volatile Transition Documents

Send process transition information in volatile mode. Enabled by default. This setting applies to Universal Messaging and Broker (deprecated) transition documents and referenced process start documents, and also to JMS transition and referenced process start messages.

- Select **Volatile Transition Documents** to specify the following:
  - **For Subscription (Publishable Documents) protocol:** Process transition documents and referenced process start documents are stored in memory.
  - **For JMS (Triggered Processes protocol):** Process transition messages and referenced process start messages are delivered to the JMS deliveryMode interface as NON-PERSISTENT.
- Clear this check box to specify the following:

- **For Subscription (Publishable Documents) protocol:** Process transition documents and referenced process start documents are stored on the local hard disk drive.
- **For JMS (Triggered Processes protocol):** Process transition messages and referenced process start messages are delivered to the JMS deliveryMode interface as PERSISTENT.

Enabling **Volatile Transition Documents** can significantly improve performance when documents are large (2 MB or larger). However, if the Universal Messaging, Broker (deprecated) or JMS server fails while a step is running, the process cannot automatically recover and completion cannot be guaranteed, and the document or message will be lost. If you are logging step pipelines to the Process Audit Log database component, you can manually recover the process by resubmitting steps through webMethods Monitor.

About Message Provider Behavior:

When the message provider receives a process transition document or referenced process document, it places the document in the process trigger's client queue and also stores it either in memory or on disk.

When the trigger retrieves the document, if the document was stored in memory it is immediately acknowledged and deleted from the message provider. If the document was stored on disk on the message provider, the document is acknowledged and deleted by the message provider after the process has published the next process transition document or the process completes.

Volatile Tracking

Store process tracking information in memory only. Disabled by default.

Note: If the Process Engine is running in a clustered environment, volatile tracking cannot be used, and this setting is ignored.

- Select **Volatile Tracking** to specify that the Process Engine stores process status in memory.
- Clear this check box to specify that the Process Engine stores process status in the Process Engine database component.

The Process Engine stores process status while a step that requires it is running. Process status is comprised of content from:

- External documents and process transition documents
- Referenced process documents

- Process and step status
- Process iteration count and correlation IDs
- Step and process timeouts

Using volatile tracking can significantly improve performance. However, if you use volatile tracking and a server fails while running a step, process status will be lost.

If you are logging process step status to the Process Audit Log database component, the step iteration count will be inaccurate in webMethods Monitor, making it harder to address the negative effects of server failure and to determine how much work has been duplicated.

If you choose to store process status in the Process Engine database component, you must configure the database component. For instructions, see "Configuring and Monitoring the Process Engine" in the PDF publication *Administering webMethods Process Engine*. For more information about the Process Engine database component, see *Installing Software AG Products*.

If you choose to store process status in the Process Engine database component, you must configure the database component. For instructions, see "Configuring and Monitoring the Process Engine " on page 33. For more information about the Process Engine database component, see *Installing Software AG Products*.

Minimum Logging Level

Sets the minimum audit logging threshold for this process at run time. Set to **5 - Process and all events, activities, and looped activities** by default.

At generation time, the **Minimum Logging Level** is set in webMethods Monitor based on this value. On subsequent generations, if the **Minimum Logging Level** is increased in Designer, the level in Monitor is also increased. If the **Minimum Logging Level** in Designer is lowered in subsequent generations, the level in Monitor is not lowered. You must explicitly lower the audit logging level in Monitor.

If a user attempts to set a new process audit logging level in Monitor, the user will not be able to specify a logging level that is numerically lower than the value you specify here. For example, if you specify a level of **2-Errors Only** here, the user will not be able to specify a logging level of **1-None** in webMethods Monitor; the user's choices are limited to audit logging levels 2, 3, 4, and 5.

If you are sending process data to webMethods Optimize to run historical data fit distributions in Process Simulation, you must

set the minimum audit logging level to **5-Process and all events, activities, and looped activities**. If you need only process-level logging and step errors to be sent to Optimize, then logging level 3 is sufficient.

- For more information about process audit logging, including descriptions of logging levels, see the PDF publication *webMethods Monitor User's Guide*.

Select one of the following values from the drop-down list:

1-None

2-Errors only

3-Process only

4-Process and start events

5-Process and all events, activities, and looped activities

Note: Logging level **6 - Process and all events, activities, and looped activities** was available in version 8.2 but has been removed in later versions and its functionality moved into logging level 5.

About Logging in webMethods Monitor

The Process Engine logs process status to the Process Audit Log database component. webMethods Monitor displays the logged data and enables you to perform a variety of actions with it.

For complete information about setting up process logging, including setting the **Logging Level** and **Log Transitions** properties, see the PDF publication *webMethods Monitor User's Guide*.

For information about the Process Audit Log database component, see the PDF publication *Installing Software AG Products*.

4 Configuring and Monitoring the Process Engine

Accessing the Process Engine Home Page

As a component that runs within the Integration Server environment, the Process Engine is configured and monitored through the webMethods Integration Server Administrator, a Web-based interface available through your local browser. The Integration Server must be running to access the Integration Server Administrator.

You can start the Integration Server Administrator from the installation menu on your system, or by entering a URL in your browser's address field that represents the name of the server the Integration Server is running on, as well as its port number (the default port number is 5555). For example: `http://localhost:5555`.

To access Process Engine home page

1. Log on to the Integration Server Administrator and click **Packages > Management**.
2. Locate the WmPRT package and click the Home Page icon  .

Note: You can click the **Logout** link at the top of the page to log out of the Process Engine home page. You will still be logged into the Integration Server Administrator. The **Logout** link terminates only the session for the Process Engine home page.

About the Process Engine Home Page

The Process Engine home page is a comprehensive view of the configuration and run-time properties of the Process Engine. It is useful as a troubleshooting tool because it provides a complete view of your Process Engine configuration in a single location.

The home page provides you with access to several pages, each containing a specific type of information, as described in the table below.

Page	Description	For more information, see
Dashboard	Indicates the status of several subsystems.	"Viewing the Process Engine Dashboard" on page 35
Startup Messages	Displays the error messages that Process Engine issued at startup.	"Viewing Process Engine Startup Messages" on page 37
Settings	Displays the Process Engine configuration settings and allows you to edit them.	"Configuring Process Engine Settings" on page 37

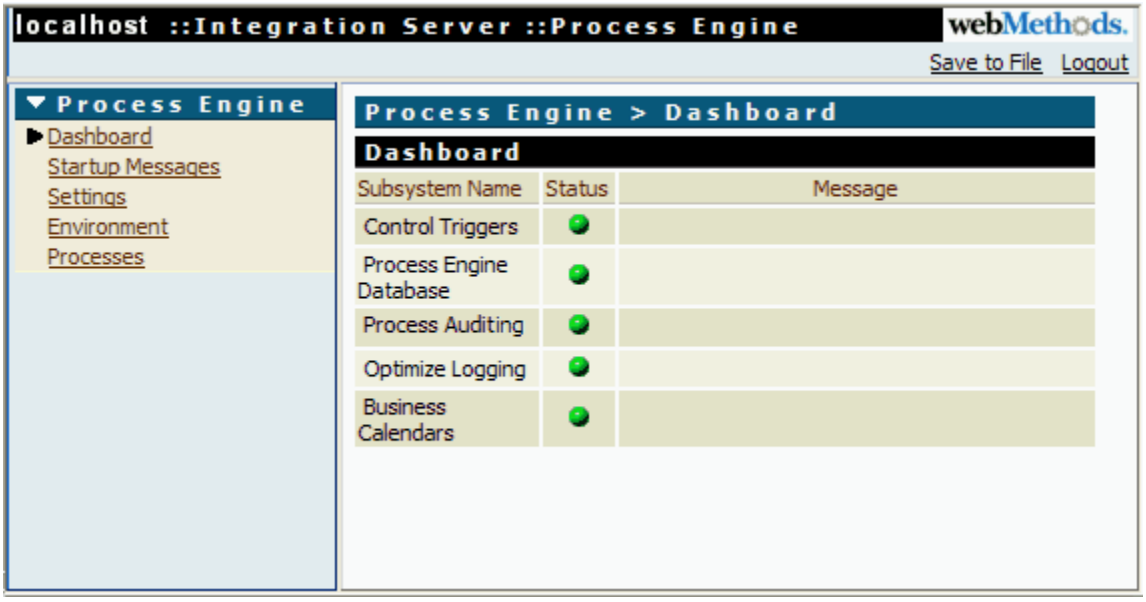
Page	Description	For more information, see
Environment	Displays information about the Process Engine environment.	"Viewing Information about the Process Engine Environment" on page 43
Processes	Displays information about the available process models.	"Viewing Process Model Information" on page 44

Viewing the Process Engine Dashboard

The Dashboard page, shown below, provides you with basic status information for Process Engine subsystems.

To view the Dashboard page

1. Display the Process Engine home page as described in "Accessing the Process Engine Home Page" on page 34.
2. Click **Dashboard**.



Icon	Status
	Running and in a normal condition.

Icon	Status
	An *optional component* that is not configured. See the **Message** column for further information.
	A *required component* that is either not configured, has been configured but is not currently running, or is configured incorrectly. See the **Message** column for further information.

3. If you encounter yellow or red status icons, refer to the other available page links to look for more information, such as **Startup Messages**, **Settings**, and **Environment**.

Additional Information About Dashboard Status

The status indicators on the Dashboard page enable you to assess the current Process Engine conditions in a compact format.

Note: In versions of Process Engine prior to v9.6, the Dashboard display provides monitoring of the Audit Logging component. In version 9.6 and later, the Audit Logging mechanism is replaced with an internal logging mechanism, and the Audit Logging component is no longer displayed in the Dashboard.

- **Control Triggers**. A green status icon indicates that the JMS control triggers and Broker (deprecated) control triggers for Process Engine are configured correctly and running. For more information, see "Viewing Information about the Process Engine Environment" on page 43.
- **Process EngineDatabase**. A green status icon indicates that the Process Engine database is connected and is configured with the correct database schema version. No check is made of the actual database structure or contents. If a schema version conflict is found, both the expected version and the found version are displayed in the **Message** column. You can update the database and obtain additional information with the webMethods Database Component Configurator. For more information, see the PDF publication *Installing Software AG Products*.
- **Process Auditing**. A green status icon indicates that:
  - The Process Audit database is connected and is configured with the correct database schema version. No check is made of the actual database structure or contents. If a schema version conflict is found, both the expected version and the found version are displayed in the **Message** column. You can update the database and obtain additional information with the webMethods Database Component Configurator. For more information, see the PDF publication *Installing Software AG Products*.
  - The Failed Process Audit cache is connected and configured. This cache temporarily retains any audit logging records that could not be written to the Process Audit database. Under normal circumstance, these records are regularly transferred to the Process Audit database and deleted from the cache. The

presence of unlogged process audit records in the cache will cause a red status icon to appear. In this case, you must determine what is preventing these records from being transferred to the Process Audit database.

- **Optimize Logging**. A green status icon indicates that the Process Engine is configured correctly to deliver logging messages to webMethods Broker (deprecated). To configure a webMethods Broker (deprecated) URL, see "Configuring Process Engine Settings" on page 37.
- **Business Calendars**. A green status icon indicates that the Process Engine is connected to My webMethods Server and is able to obtain any business calendars specified by process model timeout definitions. The status check does not verify the presence of any particular business calendar. For more information, see "Business Calendar Prerequisites" in the *webMethods BPM Process Development Help*.

Viewing Process Engine Startup Messages

The Startup Messages page enables you to view server and error log entries that the Process Engine generated during startup. The messages that Process Engine displays on this page are purged and refreshed with each Process Engine startup.

To view the startup messages

1. Display the Process Engine home page as described in "Accessing the Process Engine Home Page" on page 34.
2. Click **Startup Messages**.

Configuring Process Engine Settings

The Settings page enables you to view the current values of the Process Engine configuration settings or to modify those configuration values.

Note: The configuration settings are not automatically propagated to other Process Engine nodes in a clustered environment. You must manually apply the configuration settings to each individual Process Engine node in your environment.

To configure Process Engine properties

1. Display the Process Engine home page as described in "Accessing the Process Engine Home Page" on page 34.
2. Click **Settings**.
3. On the Settings page, click the **Edit Process Engine Settings** link.
4. On the **Process Engine > Settings > Edit** page, the settings described in the table below are available for modifications:

Property	Description
Database Operation Retry Limit	Specifies how many times the Process Engine attempts to retry a failed database operation that could succeed in a subsequent attempt. For example, a database operation to secure a step lock for a join step might fail if another Process Engine thread has already locked this same step. In this situation, the Process Engine uses the **Database Operation Retry Limit** to determine how many times the current thread should retry the operation to acquire the step lock. Under normal conditions, the thread that has the step lock will hold it only a short time before releasing it, allowing the current thread to acquire the step lock on a subsequent retry. The default value is 360.
Database Operation Retry Interval	Specifies how many seconds the Process Engine pauses before retrying a failed database operation. The default interval is 1 second.
Cleanup Service Execution Interval	Specifies how many seconds the Process Engine pauses between executions of the Storage Cleaner utility that removes data for completed and failed processes from the Process Engine database component. For more information, see "About the Process Engine Storage Cleaner" on page 14. Specify a value that will clean the database often enough to keep data from straining the database size limit, but not so often as to degrade performance. The default interval is 600 seconds. When two or more Process Engines are operated in a cluster, be sure that Storage Cleaner is enabled on at least one node. For more information, see "Special Considerations in Clustered Operation" on page 14. To disable the Storage Cleaner, set the value to 0 (zero) and reload the WmPRT package. For more information, see "Disabling/Enabling the Process Engine Storage Cleaner" on page 41.
Completed Process Expiration	Specifies the period of time that must elapse before Completed process instance data becomes eligible for deletion by the Storage Cleaner. See **Cleanup Service Execution Interval**, above. The default value is 60 seconds.

Property	Description
Failed Process Expiration	Specifies the period of time that must elapse before Failed process instance data becomes eligible for deletion by the Storage Cleaner. See **Cleanup Service Execution Interval**, above. The default value is 3600 seconds.
JMS Server URL	Specifies the URL of the JMS server used by Process Engine to publish audit messages to Optimize. This field is initially empty following installation. You can type your own URL in this field, or you can click the **Default** button at the bottom of the **Edit** page. This action sets the following values: ■ If you specified a webMethods Broker (deprecated) server, instance and port number during installation, that value is inserted here. ■ Otherwise, the default value `broker://localhost:6849/Broker #1/analysis` is inserted. To specify a webMethods Universal Messaging server, type the URL for that server. At installation, the default value is `nsp://localhost:9000`. If you have specified a different server and port number, type that information here.
Duplicate Event Detection	Indicates whether you want to enable the duplicate detection feature in Process Engine. Duplication can occur when the Integration Server is stopped abruptly after starting a process, but before acknowledging the input document. This situation applies only when using guaranteed input documents (non-volatile) and when volatile transition documents are not used. ■ *Select the check box to enable duplicate event detection*, which prevents the creation of a redundant process for a duplicate document. The Process Engine recognizes a duplicate document and determines how best to process the document. For more information about how Process Engine operates when you select this check box, see How Duplicate Detection Works. **Important:** To use the duplicate detection feature, you must also configure the generated subscription or transition trigger of the process in Software AG Designer. For more information, see "Requirements for Using the Duplicate Event Detection Feature" on page 42.

Property	Description
	■ *Clear the check box to disable duplicate event detection*; duplicate documents are sent to and processed by Process Engine normally, which could lead to duplicate processes. The default is to disable duplicate event detection.
Cache Cleanup Interval	Specifies the maximum number of seconds to hold instance complete events in cache. The Process Engine collects instance complete events in an in-memory cache. When the **Cache Cleanup Interval** elapses, or the maximum number of events is reached, as specified by the **Maximum Cache Buffer Size** setting, the Process Engine empties the cache and publishes a single bulk control event to notify other Process Engine nodes in the environment that instances have completed. Specify the number of seconds to wait before emptying the cache and publishing the bulk control event. The default is 60 seconds.
Maximum Cache Buffer Size	Specifies the maximum number of instance complete events to hold in the cache. For more information, see the **Cache Cleanup Interval** setting, above. Specify the maximum number of instance complete events to hold in cache. The default is 1000.
External Cluster	Indicates whether you want the Process Engine to operate as if it were clustered. ■ Select the check box to enable external clustering. Enable this setting if your Integration Servers are not part of a cluster, but you still want your Process Engines to act as if they are clustered. When Process Engines are in an external cluster, they share: ■ webMethods Broker (deprecated) queues, that is, they use the same webMethods Broker (deprecated) prefix. ■ Database for step locking. **Note:** When this setting is selected, any documents published to webMethods Broker (deprecated) with no subscriber

Property	Description
	will be routed to the webMethods Broker (deprecated) Dead Letter Queue (DLQ). If you do not monitor and purge the DLQ, the queue can fill to capacity and create problems. ■ Clear the check box if you do not want the Process Engines to act as if they are clustered. The default is that Process Engine is *not* part of an external cluster. **Note:** If Process Engine is running in an Integration Server that is part of a cluster, it operates in cluster mode regardless of the **External Cluster** setting.

5. Click **Submit**, then reload the WmPRT package or restart Integration Server.

Note: If at any time you need to revert to the default configuration settings, edit the Process Engine configuration properties and click **Default**.

Disabling/Enabling the Process Engine Storage Cleaner

For more information about the Storage Cleaner, see "About the Process Engine Storage Cleaner" on page 14.

To disable or enable the Storage Cleaner

1. Display the Process Engine home page as described in "Accessing the Process Engine Home Page" on page 34.
2. Click **Settings**.
3. On the Settings page, click the **Edit Process Engine Settings** link.
4. Do either of the following:
   - To disable the Storage Cleaner, type `0` (zero) in the **Cleanup Service Execution Interval** field.
   - To enable the Storage Cleaner, type an interval value in the **Cleanup Service Execution Interval** field. For more information, see "Configuring Process Engine Settings" on page 37.
5. Click **Submit**.
6. Reload the WmPRT package.

Requirements for Using the Duplicate Event Detection Feature

To use the duplicate detection feature, you must:

- Configure the Process Engine **Duplicate Event Detection** property as described in "Configuring Process Engine Settings" on page 37.
- Modify the process model's generated trigger in Software AG Designer as follows:
  - Set the **Exactly Once/Detect Duplicates** property to `true`.
  - Set the **Exactly Once/Document Resolver Service** property to wm.prt.duplicate:checkDuplicates, located in the WmPRT package.

> **Important:** After changing the **Exactly Once/Detect Duplicates** property for a model's trigger (either subscription or transition), the WmPRT package must be reloaded. To reload the package, open the webMethods Integration Server Administrator, open **Packages > Management**, and click the WmPRT reload icon in the **Reload** column.

Additionally, to use the duplicate detection feature, the process model versions must meet these requirements:

- A UUID, a unique identifier that identifies the published document, must be present in the document being published (for example: all documents published by webMethods Broker (deprecated) contain a UUID).
- All publishable documents, including process transition documents, must be non-volatile documents.
- The process must have volatile tracking disabled and volatile transition documents disabled.
- If you want to be able to resubmit a failed process, the model must log all steps and enable each step for resubmission.

> **Note:** When a process has the **Optimize Locally** option enabled, duplicate event detection is limited to the last unacknowledged document.

How Duplicate Detection Works

When you enable the duplicate detection feature, as described in "Configuring Process Engine Settings" on page 37 and "Requirements for Using the Duplicate Event Detection Feature" on page 42, webMethods Broker (deprecated), Integration Server, and Process Engine work together to detect duplicate documents to prevent the creation of a redundant process for a duplicate document.

Initially, webMethods Broker (deprecated) detects when a document is a duplicate. To enable duplicate detection, you modify the generated triggers to identify the wm.prt.duplicate:CheckDuplicates service as the service to resolve duplicate documents. As a

result, when webMethods Broker (deprecated) detects a duplicate document, it invokes the wm.prt.duplicate:checkDuplicates service.

The wm.prt.duplicate:checkDuplicates service determines how to handle the duplicate document. If the service determines that:

- **Process Engine has not yet received the document**—the service returns the NEW status, which in turn, causes the Integration Server to deliver the document as usual.
- **Process Engine has previously received the document and the Process Engine has already completed processing the document**—the service returns the DUPLICATE status, which in turn, causes the Integration Server to dispose of the duplicate document.
- **Process Engine has previously received the document and the Process Engine has started processing the document, but processing is *not* complete**—the service returns the IN DOUBT status, which in turn, causes the Integration Server to write an entry to the WMDOCUMENT table of the Process Audit database component so that a webMethods Monitor user can decide how to handle the document.

Viewing Information about the Process Engine Environment

The Environment page enables you to view settings that affect the Process Engine environment.

To view Process Engine Environment Settings

1. Display the Process Engine home page as described in "Accessing the Process Engine Home Page" on page 34.
2. Select **Environment** in the navigation on the left.

   The information from the following table is available on the **Process Engine > Environment** page:

Section	Description

General	The **General** section provides the following information about the Process Engine:
	■ **Clustered**. Indicates whether the Process Engine is operating in cluster mode. The setting is **Yes** when it *is* operating in cluster mode or **No** when it is *not* operating in cluster mode. The setting will be **Yes** when either of the following is true:
	  ■ Process Engine is running in an Integration Server that is running in a cluster.
	  ■ The External Cluster property is enabled indicating that the Process Engine is running in an external cluster. For more information about this property, see "Configuring Process Engine Settings" on page 37.

Section	Description
	■ **Performance Collection**. Indicates if the Process Engine performance collector is currently running. The performance collector collects performance data about process instances that are running. The setting is **On** when it is running or **Off** when it is not running.
JDBC Connections	The **JDBC Connections** section lists the JDBC connection properties for those JDBC pools relevant to the Process Engine at runtime. You can modify the JDBC connection information from the **Settings > JCBC Pools page of the Integration Server Administrator**.
Control Triggers	The **Control Triggers** section lists triggers that Process Engine uses across all process instances and indicates if the trigger is enabled. For additional information about these triggers and to modify them, use these webMethods Integration Server Administrator pages: ■ For webMethods Broker (deprecated) triggers: **Settings > Messaging > Broker/Local Trigger Management** ■ For JMS triggers: **Settings > Messaging > JMS Management**

Note: In versions of Process Engine prior to v9.6, the Environment page provides information about the Logger components that operate as part of the Audit Logging mechanism. In version 9.6 and later, the Audit Logging mechanism is replaced with an internal logging mechanism, and the Loggers table is no longer displayed in the Environment page.

Viewing Process Model Information

The Processes page enables you to view information about the process models currently available in the Process Engine. You can:

- Determine the enabled version of a process model.
- View process settings for each process model version.
- View a list of triggers associated with each process model version and whether a trigger is enabled or disabled.
- View information about My webMethods Server e-form listeners for process versions for which listeners are configured.

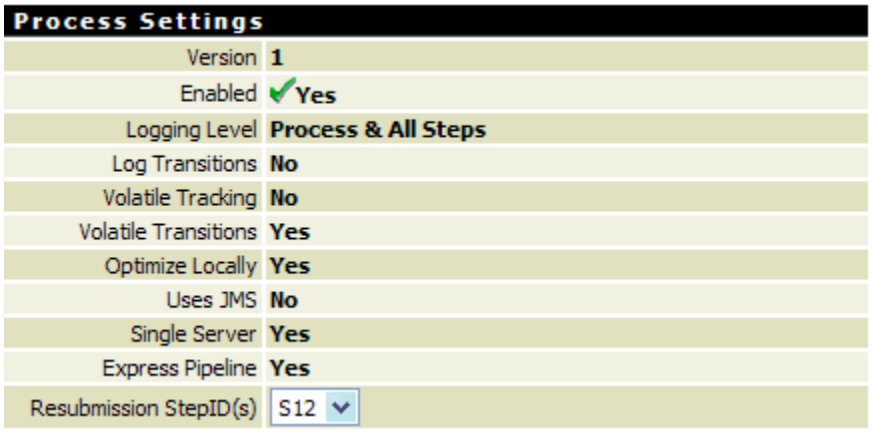
To view process model information

1. Display the Process Engine home page as described in "Accessing the Process Engine Home Page" on page 34.
2. Click **Processes**.
3. In the left pane of the **Process Engine > Processes** page, click the process model you want to examine.

   Information about the versions of the selected process model appears in the right pane. The **Enabled** column displays **Yes** to indicate an enabled version.
4. To view details of a specific process model version, click the name of version.

   The Details page provides the following information:

   - The **Process Settings** section displays settings for the selected process model version, for example:

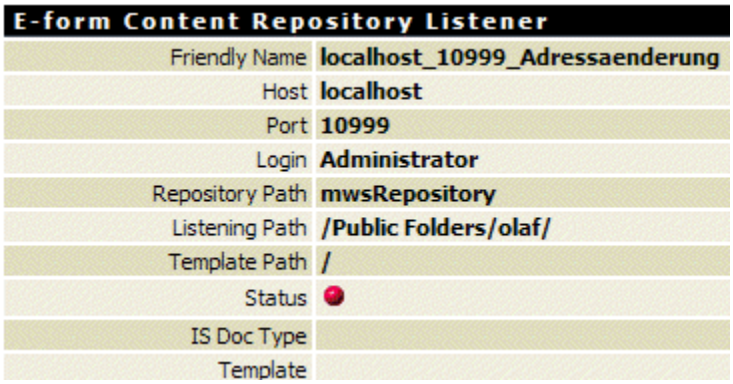


     For more information about many of these settings, see Setting Quality of Service for a Process. The steps listed in **Resubmission StepID(s)** list are the steps that have been enabled for resubmission.

   - The **Triggers** section provides a list of triggers associated with the process model version and if each trigger is enabled or disabled.
   - The **E-form Content Repository Listeners** section lists My webMethods Server listeners associated with the process model version. The Process Engine displays the **E-form Content Repository Listeners** section only if listeners are configured for the process model version.

     Click the listener name to view details about a listener:

     Note: The e-form listeners displayed here are those created with the native webMethods e-form functionality.



E-form Content Repository Listener	
Friendly Name	**localhost_10999_Adressaenderung**
Host	**localhost**
Port	**10999**
Login	**Administrator**
Repository Path	**mwsRepository**
Listening Path	**/Public Folders/olaf/**
Template Path	**/**
Status	
IS Doc Type	
Template	

The Process Engine displays a status icon to indicate whether a listener is running (status icon) or stopped (status icon). If a listener is stopped, you can restart it by clicking the **Restart** link. The Process Engine displays messages about the outcome of the restart at the top of the page.

5. Click **Return to Process Details** to leave the e-form listener details page.

Saving Process Engine Settings to a File

If you are experiencing problems and need to send Software AG Global Support information about your Process Engine, you can save the Process Engine information to an XML file.

To save Process Engine information to an XML file

1. Click **Save to File** at the top of the page.
2. Locate the XML file in the following file system location:

   Integration Server_directory\instances*instance_name* \packages\WmPRT\pub \dashboard.xml.

Configuring Process Engine to Work with Universal Messaging

This section explains how to configure Process Engine to use Software AG Universal Messaging as its JMS provider.

Universal Messaging is a message-oriented middleware product that guarantees message delivery across public, private, and wireless infrastructures. Universal Messaging overcomes the challenges of delivering data across different networks. It provides guaranteed messaging functionality without the use of a Web server or modifications to firewall policy.

Universal Messaging consists of a realm server and clients. The realm server is a Java process that is capable of delivering high data throughput to large numbers of clients.

For complete information about Universal Messaging, see the Universal Messaging documentation.

Limitations

The integration of webMethods products with Universal Messaging is subject to a number of limitations that exist between Integration Server and Universal Messaging. For specific information, see the chapter "Configuring Integration Server for JMS Messaging" in the PDF publication *webMethods Integration Server Administrator's Guide*.

Overview

Install Universal Messaging and set up your Universal Messaging environment before you begin configuring Process Engine to use Software AG Universal Messaging.

Note: Be sure to start the Universal Messaging server: **Software AG > Start Servers > Start Universal Messaging Realm Server > Start umserver**.

You must then configure your environment as described in "Configuring Process Engine for Universal Messaging ", below.

In Universal Messaging Enterprise Manager, you must also create a new connection factory.

Note: If the value of the `Message Processing/Max execution Threads` trigger property is increased to more than the default value of ten, then the value of the Universal Messaging property `ClientQueueWindow` should be increased accordingly using Universal Messaging Enterprise Manager. For example, if the trigger max execution thread is set to 20, then the client queue window size should be set to 20.

When you are finished, you can optionally review the security settings for the newly created components.

Configuring Process Engine for Universal Messaging

Note: In addition to this procedure, you must also configure Integration Server to work with Universal Messaging. For more information, see the PDF publication *webMethods Integration Server Administrator's Guide*.

For additional information, see the JNDI Integration procedures in the Universal Messaging documentation.

To configure Process Engine to use Universal Messaging

1. Start the Universal Messaging Enterprise Manager: **Software AG > Administration > Universal Messaging Administration > umserver > Enterprise Manager**.

2. In the left-hand navigation area, click **Realms > umserver**and click the **JNDI** tab.

   Note: You may have to click the **Apply** button on the **JNDI** tab to populate the **Namespace** area.

   To create a connection factory:

   a. In the **Namespace** area, right-click **Connection Factories**.

   b. Click **New Connection Factory (Shared Durable)**.

      Use the Shared Durable setting for both an Integration Server cluster and for a single Integration Server.

   c. Type `local_um` as the connection factory name.

   d. Click **OK**.

3. Review security privileges.

   Note: By default, full privileges are assigned for `Everyone`. If you want to maintain limited access, you can instead set the privileges to the OS user and system that the Integration Server is running on. For example, if Integration Server is running on server1 under OS user wmuser, set the privileges to `wmuser@server1` instead of `Everyone`.

   a. In the left-hand navigation panel, click **Realms > umserver > PEBroadcastTopic** and click the **ACL** tab.

   b. Review and modify the security privileges as required.

   c. In the left-hand navigation panel, click **Realms > umserver > PERestartTopic** and click the **ACL** tab.

   d. Review and modify the security privileges as required.

5 Collecting Process Troubleshooting Information

Introduction

When you develop a new process model, there may be times when the process does not execute as expected, and you want to find out why. Similarly, an existing process model may suddenly begin to behave in an unexpected manner, and you want to know why. The Process Engine provides you with the ability to gather troubleshooting information in the following ways:

- You can collect a package of comprehensive process information for troubleshooting purposes. For more information, see "About Process Troubleshooting Information" on page 50.
- You can collect logging information, as described in the topic "Process Logging Behavior" in the *webMethods BPM Process Development Help*.

You can use this information for your own troubleshooting efforts, and you may be asked to provide either or both of the above to Software AG Global Support if you have opened a support issue.

About Process Troubleshooting Information

The Process Engine enables you to create a zip file that creates a comprehensive package of information pertaining to a specific process instance and its process model. The generated zip file contains the following information:

- *processModelName* **.process file**. Defines the process model that a process instance is started from.
- **DB_config.txt file**. Provides the version number of the Process Engine and process audit databases in use.
- **inspectDB.xml file**. Contains the Process Engine database information for the specified process instance.
- *packageName* **.zip file**. Contains the contents of the Designer-generated process package installed in Integration Server.
- **QoS_settings.txt file**. Describes the Quality of Service settings currently specified for the process.
- **WmPRT_version.txt file**. Describes the Software AG version and build number of the Process Engine, as well as any applied fixes.
- *instanceid* **.log file**. Available when instance logging is enabled for the process model. For more information, see the topic "Process Logging Behavior" in the *webMethods BPM Process Development Help*.

The generated zip file can be created and downloaded from Integration Server Administrator, and is also available from within your Integration Server installation.

Creating a Process Troubleshooting Package

The Process Engine provides you with the ability to gather process troubleshooting information and package it into a single zip file.For more information about the contents of the file, see "About Process Troubleshooting Information" on page 50.

To create a process model troubleshooting package

1. In webMethods Monitor, obtain the process instance ID and iteration number of a process instance of the process model you want to work with. You can also obtain this information from the server.log file or, if enabled, from a process instance log file.
2. Log on to the Integration Server Administrator and click **Packages > Management**.
3. Locate the WmPRT package and click the Home Page icon  .
4. In the **Process** list, click **Diagnostic**.
5. On the **Diagnostic** page, type the process instance ID in the **Instance ID** field, and type the iteration number in the **Instance Iteration** field.
6. Click **Submit**.

   If the values entered in step 5 are valid, the zip file package is created and is available from a link on the Save to File page, and in the WmPRT\pub directory. The zip file name is in the format *processModelName_processInstanceID_processIterationID* .zip
7. Click the link to open or save the zip file, or open the zip file in the WmPRT\pub directory.

   Note: If you click another location in the WmPRT Home Page menu, the Save to File page is closed and the link is no longer available. However, you can use the browser's **Back** button to return to the Save to File page, or you can access the zip file in the WmPRT\pub directory.

6 Process Engine Services

Process Engine Built-In Services Location

The built-in services in this chapter are installed on the Integration Server as part of the WmPRT package. These Java services can be found in the indicated folder location in the Package Navigator view of Designer, or in the Package Management link in the Integration Server Administrator.

This chapter describes the services and supporting elements found in the \pub folder. You can use these services as templates to create services in Designer that perform a wide variety of actions on the services running in the Process Engine on the connected Integration Server.

For additional information about working with services in Designer, see *webMethods Service Development Help*.

Summary of Elements in the WmPRT\pub Folder

The available elements in this folder are listed in the following table:

Element	Package and Description
pub.prt:CorrelationService	WmPRT. Specification that describes the inputs and outputs required for a correlation service.
pub.prt.admin:changeProcessStatus	WmPRT. Changes the process status by broadcasting a request to all servers participating in the process.
pub.prt.admin:deleteProcess	WmPRT. Deletes information for a specific process instance from the Process Engine database.
pub.prt.admin:resumeProcesses	WmPRT. Service that allows users to resume processing for one or all previously suspended process models.
pub.prt.admin:scanPackage	WmPRT. Tells the Process Engine to scan a specified package for new or updated process model version fragments and use these fragments to update its internal model index.

Element	Package and Description
pub.prt.admin:suspendProcesses	WmPRT. Service that allows users to suspend processing for any or all available process models.
pub.prt.audit.truncateProcessAtRest	WmPRT. Truncates the Process Audit table WMPROCESSATREST. You can schedule this service to run on an interval of your choosing using the Integration Server Scheduler.
pub.prt.correlate:deleteCorrelation	WmPRT. Deletes all mappings between the specified process instance ID and any correlation IDs or conversation IDs.
pub.prt.correlate:establishCorrelation	WmPRT. Sets up a correlation between a correlation ID and a process ID or between a conversation ID (for a Trading Networks document) and a process ID.
pub.prt.correlate:lookupCorrelation	WmPRT. Returns the process instance ID that is associated with the specified correlation ID or conversation ID. If no association exists, creates a new process instance ID and mapping.
pub.prt.debugger:cleanupDebuggerTables	WmPRT. Cleans up process debugger database tables by deleting records before a given timestamp.
pub.prt:ErrorService	WmPRT. This specification has been deprecated and should no longer be used.
pub.prt.ExceptionTransitionInfo	WmPRT. This document type describes the information passed in the pipeline from a step when that step has taken one of the various Error Transitions. This document type is provided as a convenience to the Designer user to map any or all of the fields described in this document type.

Element	Package and Description
pub.prt.jms:send	WmPRT. Sends a JMS message. This service encodes an IS Document into a JMS message and sends it to the specified destination using the specified options. The main difference between this service and the pub.jms:send service in WmPublic is that this service allows the user to easily specify the type of the document, which is required by the Process Engine to kick off a process instance, as well as make it convenient to format the JMS message appropriately for use with the Process Engine.
pub.prt.log:logActivityMessages	WmPRT. Logs process activity messages to the IS Core Audit Log database.
pub.prt.log:logCustomID	WmPRT. This service associates a "friendly name" (the customID) with a Process Instance identifier. This friendly name can be used to search for the process instance in monitor.
pub.prt:ProcessData	WmPRT. Document type that describes the structure of the *ProcessData* section of the pipeline for a process.
pub.prt.SubprocessModel	WmPRT. Document type that describes the information needed to invoke a dynamic referenced process.
pub.prt.timer.process:cancel	WmPRT. This service cancels the process timer for the specified process instance.
pub.prt.timer.process:create	WmPRT. This service creates a process timer for the specified process instance.
pub.prt.timer.process:createWithBusinessCalendar	WmPRT. This service creates a process timer for the specified process instance and the specified business calendar.

Element	Package and Description
pub.prt.timer.process:createWithDate	WmPRT. This service creates a process timer for the specified process instance and the specified date.
pub.prt.timer.process:get	WmPRT. This service returns the actual date that the timer will expire for the specified process instance.
pub.prt.tn:deleteByCID	WmPRT. Deletes a process instance associated with a given conversation ID.
pub.prt.tn:getPIDforCID	WmPRT. Returns the process instance ID for a given conversation ID.
pub.prt.tn:getRoleInfo	WmPRT. Fetches role information for a specified role in process.
pub.prt.tn:handleBizDoc	WmPRT. Sends a Trading Networks BizDocEnvelope (Trading Networks document) to the Process Engine, to allow the document to be processed as part of a business process.
pub.prt.tn:mapCIDtoPID	WmPRT. Sets up a mapping between the specified conversation ID and process instance ID.
pub.prt.tn:MatchBizDoc	WmPRT. Matches the supplied business document ID to a valid process model. This service provides support for webMethods Rosetta Net in Process Engine.
pub.prt.tn:RoleInfo	WmPRT. Document type that describes information maintained for roles in a process.

pub.prt:CorrelationService

WmPRT. Specification that describes the inputs and outputs required for a correlation service.

A correlation service is associated with one or more receive steps in a process model version. The Process Engine uses the correlation service associated with a step to route IS documents published as inputs into that step to a running instance of the model, where appropriate. For more about correlation services, see "Correlation Services" in the *webMethods BPM Process Development Help*.

Input Parameters

ProcessModelID **String** ID of the process model with which this invocation of the correlation service is involved.

ProcessModelVersion **String** Version of the process model with which this invocation of the correlation service is involved.

> **Note:** Because a single correlation service can be associated with steps from more than one process model version, you can use the *ProcessModelID* and *ProcessModelVersion* to identify the process model version using the correlation service at run time.

LogicalServer **String** Name of the logical server that is associated with the step in the process model version with which this invocation of the correlation service is involved. In other words, the name of the logical server on which this correlation service is running.

Because a single correlation service can be used with steps that execute on different servers, you can use *LogicalServer* to identify a specific server at run time.

ProcessStepID **String** ID of the step in the process model version with which this invocation of the correlation service is involved (for example, `N3`).

Because a single correlation service can be associated with multiple steps in a process model version, you can use *ProcessStepID* to identify the specific step at run time.

DocumentName **String** Name of this document as used in the process model version (for example, "OrderDocument").

DocumentType **String** Name of this document type (for example, "orders.sap:OrderDocument").

Document **Document** The document.

Output Parameters

ProcessCorrelationID **String** Conditional. An abstract ID that correlates to the actual process instance ID of the running process. For example: "CUSTOMER-0003456977::ORDER-19477593-AR9-1000". All documents bound for the same instance of the process must return the same correlation ID. Similarly, correlation IDs must be unique across all process instances.

CorrelateAsTN **String** Conditional. Flag that indicates whether the correlation ID in *ProcessCorrelationID* is a conversation ID. The following values apply:

- `true` — Indicates that *ProcessCorrelationID* is a Trading Networks conversation ID.
- `false` — Default. Indicates that *ProcessCorrelationID* is not a Trading Networks conversation ID.

pub.prt.admin:changeProcessStatus

WmPRT. Changes the process status by broadcasting a request to all servers participating in the process.

Input Parameters

ProcessInstanceID **String** ID of the process instance that you want to change the status of.

ProcessIteration **String** The iteration of the process instance that you want to change the status of.

ProcessModelID **String** Optional. ID of the process model (ModelID) associated with the process you want to change the status of.

ProcessModelVersion **String** Optional. Version of the process model to change the status of.

Action

String Process action that is to be applied to the specified process instance. The following values apply:

- `SUSPEND`. Causes a process instance to temporarily stop executing.
- `RESUME`. Causes a suspended process to continue executing.
- `CANCEL`. Causes a process instance to terminate without an error condition.
- `FAIL`. Causes a process instance to terminate with an error condition.

EscalateFailure

String Conditional. Flag that indicates whether the parent process takes control of a failure in a referenced process. The following values apply:

- `true` — Default. The parent process receives notification of the failure from a referenced process and continues executing. The failed referenced process cannot be resubmitted as the parent process is no longer waiting for a response.
- `false` — The parent process does not receive notification of the failure from a referenced process and continues to wait for a response; this enables the referenced process to be resubmitted for another attempt at normal process completion. See the Usage Notes for more information.

Output Parameters

None.

Note: If this service runs to completion, it means that the request for a status change has been made. The servers involved handle these requests asynchronously. You can track the status of a process using webMethods Monitor.

Usage Notes

The following table shows all valid status change combinations. All combinations that are not listed in the table are invalid.

Status	Action	New Status
Started	`SUSPEND`	Suspended
Started	`FAIL`	Failed

Status	Action	New Status
Started	CANCEL	Canceled
Suspended	FAIL	Failed
Suspended	CANCEL	Canceled
Suspended	RESUME	Resumed/Started
Completed	FAIL	Failed
Failed	CANCEL	Canceled

The *EscalateFailure* parameter enables the developer to determine if a referenced process is recoverable by resubmittal after an error occurs. Prior to version 8.0 SP1, the failure of a referenced process was always escalated to the parent process. In this case, the parent process would continue execution, and the failed referenced process cannot be successfully resubmitted because the parent process is no longer waiting for a response.

For backward compatibility, the *EscalateFailure* parameter is set to true by default, replicating this behavior. However, when pub.prt.admin:changeProcessStatus is invoked and the *EscalateFailure* parameter is set to false, the failure of the referenced process is not escalated to the parent process, and the parent process continues to wait for a response. This allows the failed referenced process to be resubmitted, and upon successful completion, the referenced process will rejoin the parent process.

To implement this scenario, you must design the referenced process so that it will call pub.prt.admin:changeProcessStatus (on a terminate step, for example) with the *EscalateFailure* parameter set to false. Then, if the referenced process fails, the parent process will remain as "started", and you can then resubmit the referenced process. If the referenced process fails again, the same behavior occurs. If the referenced process completes, it will rejoin the parent instance.

A referenced process failure that occurs by a cause other than the changeProcessStatus service (for example, an error occurs and there is no error handler step) is not affected by this parameter and will continue to escalate the failure to the parent instance.

Note: To be able to resubmit the failed referenced process, you must enable the failed step for resubmission in webMethods Monitor. For more information, see the *webMethods Monitor User's Guide*.

pub.prt.admin:deleteProcess

WmPRT. Deletes information for a specific process instance from the Process Engine database.

Use this service to delete the information for a specific process instance from the Process Engine database. For example, if that process that has become unresponsive or has entered an indeterminate state. The information is removed from the Process Engine database only. The Process Audit database remains unaffected, and the process instance will continue to appear in webMethods Monitor. If you want to retain the process instance information but remove mappings to correlation IDs, use the pub.prt.correlate:deleteCorrelation service.

Input Parameters

ProcessInstanceID	**String** ID of the process instance for which you want to delete information.
ProcessIteration	**String** The iteration of the process instance that you want to delete from the Process Engine database.

Output Parameters

success	**String** Flag indicating whether the process instance information was deleted. The following values apply: ■ `true` — The process instance information was deleted. ■ `false` —The process instance information was not deleted.

Usage Notes

In normal operation, the Process Engine Storage Cleaner utility removes older process instance information from the Process Engine database at regularly scheduled intervals. You can set the Storage Cleaner intervals on the Settings page of the Process Engine home page in the Integration Server Administrator. For more information, see the topic "Configuring Process Engine Settings" in Chapter 4, "Configuring and Monitoring the Process Engine" . You can use the pub.prt.admin:deleteProcess service to remove this information outside of the regularly scheduled Storage Cleaner operation. Note that the Storage Cleaner removes process instance data for completed and failed instances that are eligible for deletion, and it also removes correlation data. For more information, see "About the Process Engine Storage Cleaner" in Chapter 1, "Concepts" .

See Also

on page 83"pub.prt.tn:deleteByCID"

pub.prt.admin:resumeProcesses

WmPRT. Service that allows users to resume processing for one or all previously suspended process models.

The service can be invoked either on a per-model basis or for all available models on a given Process Engine server. When this service is executed, the subscription triggers for the qualifying models are enabled, facilitating the creation of new instances. In addition, any previously suspended instances are set to status = RESUMED. In the case of a cluster or a distributed Process Engine set up, the node that the service is invoked on is considered primary and it broadcasts the resume action across all participating nodes that it is aware of.

Input Parameters

ProcessModelID **String** Optional. This is the process key for the model that the user wants to resume. All versions of a given model are affected.

resumeAll **String** Optional. Indicates whether or not the resume action will affect all models or only the model that is specified as the value for the parameter *ProcessModelID* . The following values apply:

- `true` — The *ProcessModedID* value, if provided, is ignored and the resume action is applied to all available models, their corresponding process instances, and their subscription triggers.
- `false` — Default. The resume action is applied only to the process model defined in the parameter *ProcessModelID* (all process instances and subscription triggers).

Output Parameters

message **String** Optional. Displays any exceptions encountered during the service call.

success **String** Optional. Returns one of the following:

- `error` — Indicates that one or more exceptions occurred during the service call.
- `true` — Indicates that the service call executed without any exceptions. It does *not* indicate that all internal tasks initiated by the service call have run to completion. To verify that the resume action is complete, refer to the Usage Notes section.

Usage Notes

This service is intended for use during scheduled maintenance periods that require processing to be paused for a period of time and then subsequently resumed. The service may be used to resume processing by either resuming a single process model or all available models on the system that were previously suspended.

The service affects all versions of targeted process models, and in the case of a cluster or distributed set up, the resume action is relayed to all participating nodes. To resume only a specific model, users can invoke this service and provide the appropriate value for the *ProcessModelID* input parameter. To resume all models on a given system, invoke the service by setting the value of the *resumeAll* parameter to `true`.

The service resumes subscription triggers for all targeted services. Any qualifying instances that were previously set to suspended by other means (for example, manually with webMethods Monitor) will be resumed, as the invocation of this service affects all qualifying instances without exception.

Note: Be aware that the complementary suspendProcesses service suspends subscription triggers for all targeted process models; during the suspension period, any incoming documents directed to those models will be cached until such time as the model is resumed. Applying the resumeProcesses service will enable the subscription triggers for all targeted models, and all cached documents will be processed. This could cause a temporarily heavy load on the system.

This service may not work as expected for environments or process models that use volatile tracking. In this case, the resume feature relies solely on the Process Engine cache in RAM to determine the instances that will be affected. However, due to its dynamic nature, the Process Engine cache may be momentarily out-of-sync, especially following a server restart or a package re-load. In such cases, resuming at the instance-level using webMethods Monitor may be the only alternative.

The resume action is a series of independent tasks across all participating nodes. To verify that the resume action is indeed complete, the users are advised to use the following actions:

- Check the Integration Server error log to make sure there were no exceptions.
- Use webMethods Monitor to verify that the status of all the qualifying instances are set to RESUMED.
- Use the webMethods Broker (deprecated) trigger management page (**Settings > Messaging > Broker/Local Trigger Management**) in the IS Administrator interface to verify that all the qualifying subscription triggers are enabled.

See Also

on page 66“pub.prt.admin:suspendProcesses”

pub.prt.admin:scanPackage

WmPRT. Tells the Process Engine to scan a specified package for new or updated process model version fragments and use these fragments to update its internal model index.

This service updates the Process Engine index as follows:

- If a fragment file is new, scanPackage adds information from this file to the Process Engine index.
- If a fragment file is modified, scanPackage replaces existing index information to reflect the modifications.
- If a fragment file no longer exists, scanPackage deletes the corresponding index information.

Input Parameters

Package	**String** Name of the package that you want to scan. If the named package does not exist on the Integration Server, information about any fragments previously loaded from that package will be deleted from the Process Engine model index.

Output Parameters

ExistingFragments	**String List** Conditional. Number of fragments contained in the package that were already known to the Process Engine.
ModifiedFragments	**String List** Conditional. Number of fragments contained in the package that had been modified since the Process Engine last read them.
MissingFragments	**String List** Conditional. Number of fragments contained in the package that have been deleted since the Process Engine last read them.
NewFragments	**String List** Conditional. Number of fragments contained in the package that are new since the Process Engine last scanned the named package.

Usage Notes

Use this service before or after replicating a package that contains Process Engine process model version fragments to accomplish the following:

- After unzipping a package onto a new server but before enabling it, invoke this service to force the Process Engine to pick up the new fragments.
- After disabling all related process model versions, zipping a package, and deleting the package from an old server, invoke this service to force the Process Engine to discard information about the old model version.

pub.prt.admin:suspendProcesses

WmPRT. Service that enables users to suspend processing for any or all available process models.

The service can be invoked either on a per-model basis or for all available models on a given Process Engine server. When this service is executed, the subscription triggers for the qualifying models are suspended and no new instances are created. In addition, any currently running instances are set to status = SUSPENDED. In the case of a cluster or a distributed Process Engine set up, the node that the service is invoked on is considered primary and it broadcasts the suspend action across all participating nodes that it is aware of.

Input Parameters

ProcessModelID **String** Optional. This is the process key for the model that the user wants to suspend. All versions of a given model are affected.

suspendAll **String** Optional. Indicates whether or not the suspend action will affect all models or only the model that is specified as the value for the parameter *ProcessModelID* . The following values apply:

- `true` — The *ProcessModelID* value, if provided, is ignored and the suspend action is applied to all available models, their corresponding process instances, and their subscription triggers.
- `false` — Default. The suspend action is applied only to the process model defined in the parameter *ProcessModelID* (all process instances and subscription triggers).

Output Parameters

message **String** Optional. Displays any exceptions encountered during the service call.

success **String** Optional. Returns one of the following:

- `error` — Indicates that one or more exceptions occurred during the service call.
- `true` — Indicates that the service call executed without any exceptions. It does *not* indicate that all internal tasks initiated by the service call have run to completion. To verify that the suspend action is complete, refer to the Usage Notes section.

Usage Notes

This service is intended for use during scheduled maintenance periods that require processing to be paused for some duration of time. The service may be used to suspend processing in bulk, impacting either a single process model or all available models on the system.

The service affects all versions of targeted process models, and in the case of a cluster or distributed set up, the suspend action is relayed to all participating nodes. To suspend only a specific model, users can invoke this service and provide the appropriate value for the *ProcessModelID* input parameter. To suspend all models on a given system, invoke the service by setting the value of the *suspendAll* parameter to `true`.

The service suspends subscription triggers for all targeted services. Note that when the triggers are suspended, the change to the trigger state is persisted through subsequent server restarts. Transition triggers are not suspended, and some preliminary transition trigger processing may occur as the result of incoming documents. Incoming documents directed to suspended models will be cached until such time as the model is resumed.

This service may not work as expected for environments or process models that use volatile tracking. In this case, the suspend feature relies solely on the Process Engine cache in RAM to determine the instances that will be affected. However, due to its dynamic nature, the Process Engine cache may be momentarily out-of-sync, especially following a server restart or a package re-load. In such cases, suspending at the instance-level using webMethods Monitor may be the only alternative.

The suspend action is a series of independent tasks across all participating nodes. To verify that the suspend action is indeed complete, the users are advised to use the following actions:

- Check the Integration Server error log to make sure there were no exceptions.
- Use webMethods Monitor to verify that the status of all the qualifying instances are set to SUSPENDED.
- Use the webMethods Broker (deprecated) trigger management page (**Settings > Messaging > Broker/Local Trigger Management**) in the IS Administrator interface to verify that all the qualifying subscription triggers are enabled.
- Use the IS Administrator Service Usage page (**Server > Service Usage**) to ensure that there are no currently running threads. This may happen if there are any long-running services that are waiting to finish.

See Also

on page 62“pub.prt.admin:resumeProcesses”

pub.prt.audit.truncateProcessAtRest

WmPRT. Truncates the Process Audit table WMPROCESSATREST. You can schedule this service to run on an interval of your choosing using the Integration Server

Scheduler. For more information, see the topic "Scheduling Services" in the PDF publication *webMethods Integration Server Administrator's Guide*.

This service is relevant for users who are implementing the optional database partitioning scripts for Process Audit data archiving. In that case, the ProcessAtRest table contains an entry for every completed process instance which, at some point, must be cleaned up. Use this service for that purpose only.

If you are not using database partitioning for Process Audit Archive, this service is not relevant as the table will contain 0 entries by definition.

Input Parameters

None.

Output Parameters

success **String** Flag indicating whether the table was truncated. The following values apply:

- `true` — The table was truncated.
- `false` — No truncation occurred.

pub.prt.CallActivityModel

WmPRT. Document type that describes the information needed to dynamically invoke one or more callable processes.

Important: This document type is used with callable processes only. If you are working with the deprecated ability to dynamically invoke a referenced processes from a call activity step, see "pub.prt.SubprocessModel" on page 78 for more information.

Input Parameters

WaitForCallActivity **String** Flag indicating whether to wait for the child process (as done with statically invoked process), or to launch it asynchronously and not expect any return documents. Applies to all process instances started from the document type.

- `true` — Default. Wait for the child process.
- `false` — Start the child process asynchronously and do not expect any return documents.

CallActivityModelID **String** The identifier of the callable process model in the format: Project/Process.

OutputPipelineName	**String** The name of the output pipeline to be returned back from each child process. No value is needed here if a value of false is used for *WaitForCallActivity* .
InputPipelines	**Document list** A list of instances that are to be started for the specified callable process model. For example, if you have 10 line items to process, there will be 10 documents in this list.

Output Parameters

None.

Usage Notes

To enable successful execution of a dynamically invoked callable process, you must ensure that these values are in the process pipeline prior to any activity (for example, input data mapping) that applies to the call activity step.

Suppose you have specified *OutputPipelineName* and a *WaitForCallActivity* value of `true`. In this case, the Process Engine waits for all instances of the child process to complete and populates the pipeline with data from each child process instance, obtained from the returned document(s). This data is now available for use.

In another example, suppose that three instances of the child process LineItem are started, and each of these instances is expected to return an output pipeline of the name LineItemPrice. In this case, the LineItemPrice data is added to the pipeline and the data in it is available for use. The order of the documents in the list is the same as the order of invocation. This enables easy location of the data in the pipeline by using the *OutputPipelineName* value.

pub.prt.correlate:deleteCorrelation

WmPRT. Deletes all mappings between the specified process instance ID and any correlation IDs or conversation IDs.

Input Parameters

ProcessInstanceID	**String** Process instance ID for the mapping(s) you want to delete.

Output Parameters

success	**String** Flag indicating whether any mappings were deleted. The following values apply: ■ `true` — One or more mappings were deleted.

- `false` — No mappings were deleted.

Usage Notes

Use this service with care. Deleting correlation mappings for running process instances could have unpredictable results.

pub.prt.correlate:establishCorrelation

WmPRT. Sets up a correlation between a correlation ID and a process ID or between a conversation ID (for a Trading Networks document) and a process ID.

Input Parameters

ProcessCorrelationID **String** The correlation ID or conversation ID that you want to map to the specified process instance ID.

ProcessInstanceID **String** Process instance ID that you want to map to the specified correlation ID or conversation ID.

MappingType **String** Optional. Flag indicating the type of ID that you supplied.

- `IS` — Default. This is a correlation ID.
- `TN` — This is a Trading Networks conversation ID.

Output Parameters

success **String** Flag indicating whether the mapping took place. The following values apply:

- `true` — The mapping was successfully established.
- `false` — The mapping could not be established.

Usage Notes

The Process Engine automatically establishes these mappings when Trading Networks or IS documents are used to start processes, or when a Trading Networks document is output from a running process. Use this service when there is no way for the Process Engine to determine the mapping itself (for example, when a process starts with an IS document and then waits for a Trading Networks document).

Use this service with care. Be sure to create correct mappings; an invalid mapping could prevent other processes from completing successfully.

pub.prt.correlate:lookupCorrelation

WmPRT. Returns the process instance ID that is associated with the specified correlation ID or conversation ID. If no association exists, creates a new process instance ID and mapping.

Input Parameters

ProcessCorrelationID **String** Correlation ID or conversation ID for which you want to return the process instance ID.

MappingType **String** Optional. Flag indicating whether *ProcessCorrelationID* specifies a correlation ID or a conversation ID. The following values apply:

- `IS` — Default. *ProcessCorrelationID* is a correlation ID for an IS document.
- `TN` — *ProcessCorrelationID* is a Trading Networks conversation ID for a Trading Networks document.

Output Parameters

ProcessInstanceID **String** Conditional. The process instance ID mapped to the specified correlation ID or conversation ID (if any).

success **String** Flag indicating whether a process instance ID was returned. The following values apply:

- `true` — A process instance ID was found or created and returned.
- `false`— A process instance ID was not returned because the correlation ID or conversation ID was not established.

Usage Notes

Use this service to check on mappings that were established with previous calls to pub.prt.correlate:establishCorrelation or, under certain circumstances, to check on the existence of a process with a particular correlation ID or conversation ID.

See Also

on page 70“pub.prt.correlate:establishCorrelation”

pub.prt.debugger:cleanupDebuggerTables

WmPRT. This service cleans up process debugger database tables by deleting records before a given timestamp.

Process Debug database records are deleted when the session ends or a new session is started. However, sometimes in the case of failures, such as the package being reloaded during debugging, data can be left in the database tables. This service can be used to delete all old records based on a timestamp provided by the user.

Input Parameters

beforeDate	**String** Optional. Date prior to which to delete Process Debug records. For example, 05/10/12. The *beforeDate* parameter requires the use of the *pattern* parameter. The current timestamp is used to populate this field if no value is specified. This results in the deletion of all Process Debug records created prior to running this service.
pattern	**String** Optional. Format of *beforeDate* value. For example, dd/MM/yy. The *pattern* parameter requires the use of the *beforeDate* parameter. This service uses the Java SimpleDateFormat class to parse dates.
daysBefore	**String** Optional. Number of days prior to today. For example, 4.

Output Parameters

None.

Usage Notes

This service can be scheduled using a wrapper service or run manually. Do not run this service while debug sessions are active. Allow any active debug sessions to complete before running this service.

If the service cannot parse the date, it does not delete any records, and displays the following message: `Parameters supplied could not be formed into a date for deletion.`

The service displays a message upon successful deletion of records: `Database tables were deleted before`*`date`*. The date is localized and uses the following format: `MM dd yyyy HH:mm:ss a`. For example: `Aug 12, 2012 12:00:00 AM`.

pub.prt:ErrorService

WmPRT. This specification has been deprecated and should no longer be used.

pub.prt.ExceptionTransitionInfo

WmPRT. This document type describes the information passed in the pipeline from a step when that step has taken one of the various Error Transitions. This document type is provided as a convenience to the Designer user to map any or all of the fields described in this document type.

Input Parameters

ErrorMessage	**String** The error message describing the reason for the error transition.
SourceStepID	**String** The Step ID of the step that encountered the error
SourceStepIteration	**String** The Step iteration count of the step that encountered the error
ExceptionType	**String** One of the following: Cancel, UnsatisfiedJoin, RetriesExceeded, ProcessTimeout, JoinTimeout, StepTimeout, or StepError

Output Parameters

None.

pub.prt.jms:send

WmPRT. Sends a JMS message. This service encodes an IS Document into a JMS message and sends it to the specified destination using the specified options. The main difference between this service and the pub.jms:send service in WmPublic is that this service allows the user to easily specify the type of the document, which is required by the Process Engine to kick off a process instance, as well as make it convenient to format the JMS message appropriately for use with the Process Engine.

Input Parameters

connectionAliasName	**String** Set this value to define the connection alias you want to use. The default PE_NONTRANSACTIONAL_ALIAS is used if no value is specified.
destinationName	**String** Set this parameter to the value of the "Destination Name" in the Subscription trigger generated by Designer for the process model that this JMS message should start. It will be similar to "YourProjectName_YourProcessName_SUBQUEUE".
destinationType	**String** Set this parameter to the value of the "Destination Type" in the Subscription trigger generated by Designer for the process model that this JMS message should start. It will usually be "QUEUE".
deliveryMode	**String** Set this to PERSISTENT or NON_PERSISTENT. This field is mapped directly to the JMSMessage.header.deliveryMode field in the pub.jms:send service in WmPublic. For more information about the pub.jms:send service see the *webMethods Integration Server Built-In Services Reference*.
priority	**String** This field is mapped directly to the JMSMessage.header.priority field in the pub.jms:send service in WmPublic. For more information about the pub.jms:send service see the *webMethods Integration Server Built-In Services Reference*.
timeToLive	**String** This field is mapped directly to the JMSMessage.header.timeToLive field in the pub.jms:send service in WmPublic. Refer to the *webMethods Integration Server Built-In Services Reference* for more information about that field.
data	**Document** Set this parameter to the IS Document that should be send in the body of the JMS message. This will be the document that actually kicks off the process instance.
documentType	**String** Set this parameter to the fully qualified name of the document type that was used for the "data" parameter. This will be the same document type that was specified in Designer as the Receive Document type.

useCSQ	**String** Set this field to `false` if guaranteed transitions are required. Otherwise set it to "true" to utilize client-side queuing.

Output Parameters

JMSTimestamp	**String** Indicates when the JMS provider sent the message
JMSMessageID	**String** Handle to lock object and not drop from pipeline.

Usage Notes:

This service is used to initiate a process instance. If you do not specify a connection alias for the *connectionAliasName* parameter, the JMS connection alias PE_NONTRANSACTIONAL_ALIAS is used by default to connect to a JMS provider and send the JMS message. You can specify a different connection alias with the *connectionAliasName* parameter. The connection alias must exist and be properly configured by an IS administrator.

This service is a thin wrapper on top of the pub.jms:send service in WmPublic, but that service may also be used to initiate a process instance. Most of the parameters specified above map directly to similarly named parameters in the WmPublic service, with the following exception:

- The pub.jms:send service has no dedicated input parameter with which to set the *documentType* value that is required by the Process Engine to correctly map a JMS message to the correct process model. Instead, callers of pub.jms:send must instantiate a *documentType* field in the *properties* document, and set the value of that field to the fully qualified name of the IS document expected by the desired process model.

  For more information about the pub.jms:send service see the *webMethods Integration Server Built-In Services Reference*.

pub.prt.log:logActivityMessages

WmPRT. Logs process activity messages to the IS Core Audit Log database.

Note: This service does not reference the logging level set by the user in webMethods Monitor, so all pertinent information is logged regardless of the logging level setting.

Input Parameters

FullMessage	**String** Optional. Complete message to record in the IS Core Audit Log database. The message can be up to 1024 bytes.

BriefMessage **String** Optional. Shortened version of the full message. The message can be up to 240 bytes.

EntryType **String** Flag indicating the type of message. The following values apply:

- `Message` — Indicates that the message is informational and no action is needed.
- `Warning` — Indicates that the message is a warning message. The process can complete successfully even if the circumstance causing the warning is not addressed.
- `Error` — Default. Indicates that the message is an error message. The process cannot complete successfully until the circumstance causing the error is resolved.

Output Parameters

None.

Usage Notes

This service can be added either to the flow service generated for a process step or to services called within that step. The service logs the input parameters to the PRA_STEP_MESSAGE log file in the IS Core Audit Log database.

Logged activity messages can be viewed in webMethods Monitor on the **Process Instance Status** and **Service Details** pages. For more information about viewing activity messages in webMethods Monitor, see the webMethods Monitor documentation.

pub.prt.log:logCustomID

WmPRT. This service associates a "friendly name" (the customID) with a Process Instance identifier. This friendly name can be used to search for the process instance in webMethods Monitor.

Note: This service does not reference the logging level set by the user in webMethods Monitor, so all pertinent information is logged regardless of the logging level setting.

Input Parameters

ProcessInstanceID **String** Process Instance ID to be associated with the customID.

customID **String** The "friendly name" of the Process Instance you wish to associate with the ProcessInstanceID.

Note: The use of the characters "&" and "="are restricted in this parameter. For example, if you create a custom ID with a format of `<fieldname1>=<valuename1>` or `<fieldname1>=<valuename1>&<fieldname2>=<valuename2>`, then Monitor will create a column for each field name and display the value of the value name in that column.

Output Parameters

None.

pub.prt:ProcessData

WmPRT. Document type that describes the structure of the *ProcessData* section of the pipeline for a process.

This pipeline data is automatically filled in by the Process Engine for every step of a process. The service for that step is then executed with this data.

Parameters

ProcessInstanceID	**String** Process instance ID of the running process.
ProcessIteration	**String** Number of times the process has been restarted (that is, iteration count).
ProcessModelID	**String** ID of the process model used by the running process.
ProcessModelVersion	**String** Version of the process model used by the running process.
ProcessStepID	**String** ID of the running step in the process.
LogicalServer	**String** Name of the logical server on which the running step was assigned and is executing.
TryCount	**String** The current iteration of the step.
LoopCounter	**String** The number of completed loops. After each loop is executed, the value of this field is incremented.
AuditContext	**String**

Roles **Document** Conditional. If this process involves Trading Networks, this will contain information about the roles in the process. The key will be the role name, and the value will be an instance of the pub.prt.tn:RoleInfo IS document type.

See Also

on page 88"pub.prt.tn:RoleInfo"

pub.prt.SubprocessModel

WmPRT. Document type that describes the information needed to dynamically invoke a referenced process.

Important: This document type is used with the deprecated ability to dynamically invoke a referenced processes from a call activity step. If you are working with dynamically invoked callable processes, see "pub.prt.SubprocessModel" on page 78 for more information.

Input Parameters

WaitForSubprocess **String** Flag indicating whether to wait for the child process (as done with statically invoked referenced processes) or to launch it asynchronously and not expect any return documents. Applies to all process instances started from the document type.

- `true` — Default. Wait for the child process.
- `false` — Start the child process asynchronously and do not expect any return documents.

SubprocessModelID **String** The identifier of the referenced process model in the format: Project/Process.

ReturnDocuments **String List** A list of document types that the parent process expects back from each child process. No value is needed here if a value of false is used for *WaitForSubprocess* .

SubprocessInstances **Document list** A list of instances that are to be started for the specified referenced process model. For example, if you have 10 line items to process, there will be 10 documents in this list.

Key	Description

Inputs **Document list** A list of document types that are needed to invoke the specified referenced process model. For example, if your model requires a LineItem document and a Customer document, you will need two entries in this list for that instance.

Key	Description
Type	**String** The fully qualified type of the document (for example, "MyPackage.docs:MyDocumentType").
Document	**Document** An input document for the given subprocess instance, as defined by the *Type* key.

Output Parameters

None.

Usage Notes

To enable successful execution of the dynamically invoked referenced process, you must ensure that these values are in the process pipeline prior to any activity (for example, input data mapping) that applies to the referenced step.

Suppose you have specified *ReturnDocuments* and a *WaitForSubprocess* value of `true`. In this case, the Process Engine waits for all instances of the child process to complete and populates the pipeline with data from each child process instance, obtained from the returned document(s). This data is now available for use.

In another example, suppose that three instances of a LineItem child process are started, and each of these instances is expected to return a LineItemPrice document. In this case, a document list named LineItemPrice is added to the pipeline and the data in it is available for use; the order of the documents in the list is the same as the order of invocation. This enables easy location of the data in the pipeline by using the return document name.

pub.prt.timer.process:cancel

WmPRT. This service cancels the process timer for the specified process instance.

Input Parameters

ProcessInstanceID	**String** Process instance ID of the process timer you want to cancel.

Output Parameters

None.

pub.prt.timer.process:create

WmPRT. This service creates a process timer for the specified process instance.

Input Parameters

ProcessInstanceID	**String** Process instance ID of the process for which you are creating the timer.
BaseDate	**String** The basis for the process timer. The following values apply: ■ `InstanceStart` — Sets the timer relative to the start time of the process instance. ■ `CurrentTime` — Sets the timer relative to the current time. ■ `ExistingTimeout` — Sets the timer relative to the expiration time of the current timer that exists for the process instance.
Days	**String** The number of days to be applied to the timer expressed as a whole number.
Hours	**String** The number of hours to be applied to the timer expressed as a whole number.
Minutes	**String** The number of minutes to be applied to the timer expressed as a whole number.
Seconds	**String** The number of seconds to be applied to the timer expressed as a whole number.
Milliseconds	**String** The number of milliseconds to be applied to the timer expressed as a whole number.

Output Parameters

None.

Usage Notes

The timer is expressed in terms of days, hours, minutes, seconds, and milliseconds. A blank value is assumed to be 0 (zero). If a *BaseDate* of `ExistingTimeout` is specified and there is no existing timer for the process, the current time will be used as the basis for the timer (same as *BaseDate* of `CurrentTime`).

pub.prt.timer.process:createWithBusinessCalendar

WmPRT. This service creates a process timer for the specified process instance and the specified business calendar.

Input Parameters

ProcessInstanceID **String** Process instance ID of the process for which you are creating the timer.

BaseDate **String** The timer start date, defined by entering one of the following string values:

- `InstanceStart` — Sets the timer relative to the start time of the process instance.
- `CurrentTime` — Sets the timer relative to the current time.
- `ExistingTimeout` — Sets the timer relative to the expiration time of the current timer that exists for the Process Instance. If `ExistingTimer` is specified and there is no existing timer for the process, the current time will be used as the basis for the timer (same behavior as specifying `CurrentTime`).

In all cases, the actual timer expiration is created based on the specified business calendar.

BusinessCalendar **String** The alias name of the business calendar in My webMethods Server that you want to reference.

Days **String** The number of days to be applied to the timer expressed as a whole number. If you do not specify days, or if you specify an invalid value, the value is assumed to be 0 (zero).

Hours **String** The number of hours to be applied to the timer expressed as a whole number. If you do not specify hours, or

	if you specify an invalid value, the value is assumed to be 0 (zero).
Minutes	**String** The number of minutes to be applied to the timer expressed as a whole number. If you do not specify minutes, or if you specify an invalid value, the value is assumed to be 0 (zero).

Output Parameters

None.

Usage Notes

The timer is expressed in terms of days, hours, minutes, seconds, and milliseconds. A blank value is assumed to be 0 (zero). If a *BaseDate* of `ExistingTimer` is specified and there is no existing timer for the process, the current time will be used as the basis for the timer (same as *BaseDate* of `CurrentTime`).

pub.prt.timer.process:createWithDate

WmPRT. This service creates a process timer for the specified process instance and the specified date.

Input Parameters

ProcessInstanceID	**String** Process instance ID of the process for which you are creating the timer.
Date	**Object** The actual date/time the timer will expire.

Output Parameters

None.

pub.prt.timer.process:get

WmPRT. This service returns the actual date that the timer will expire for the specified process instance.

Input Parameters

ProcessInstanceID	**String** Process instance ID for the process timer you are retrieving.

Output Parameters

ProcessTimeout	**Object** The actual date the process timer is set to expire.

pub.prt.tn:deleteByCID

WmPRT. Deletes a process instance associated with a given conversation ID.

Use this service to delete process state for processes that involve webMethods Trading Networks and for which you have a conversation ID rather than a process ID.

Input Parameters

ConversationID	**String** Conversation ID for the process instance for which you want to delete process state information.

Output Parameters

success	**String** Flag indicating whether the process state was deleted. The following values apply: ■ `true` — The process state was deleted. ■ `false` — The process state was not deleted.

Usage Notes

This service invokes pub.prt.correlate:lookupCorrelation to look up the mapped *ProcessInstanceID* and invokes pub.prt.admin:deleteProcess to delete the process state information.

Using this service to delete state of a running process will produce unpredictable results.

See Also

on page 61“pub.prt.admin:deleteProcess”

pub.prt.tn:getPIDforCID

WmPRT. Returns the process instance ID for a given conversation ID.

Use this service within a process that involves webMethods Trading Networks when you have a conversation ID but need the corresponding process instance ID. This service is a wrapper around pub.prt.correlate:lookupCorrelation.

Input Parameters

ConversationID	**String** Conversation ID for which you want the associated process instance ID.

Output Parameters

ProcessInstanceID	**String** Conditional. The process instance ID related to the specified conversation ID (if there is one).
success	**String** Flag indicating whether the process ID was retrieved. The following values apply: ■ `true` — The process ID was retrieved. ■ `false` — The process ID was not retrieved.

See Also

on page 86"pub.prt.tn:mapCIDtoPID"

pub.prt.tn:getRoleInfo

WmPRT. Fetches role information for a specified role in process.

Use this service within processes that involve webMethods Trading Networks.

The returned information includes the internal ID of the partner within the Trading Networks system, which you can use to retrieve the Trading Networks profile information.

Input Parameters

ProcessData	**Document** The *ProcessData* portion of the pipeline, which is standard information available for all processes. The structure of this document (IData object) is defined by pub.prt.admin:changeProcessStatus.
roleName	**String** Name of the role for which you want to retrieve information.

Output Parameters

roleInfo	**Document** Conditional. Role information that is currently available for the specified role. The structure of this document (IData object) is defined by pub.prt.tn:RoleInfo. This parameter is not present if no documents have been sent to or received for this role.

See Also

on page 88"pub.prt.tn:RoleInfo"

on page 59"pub.prt.admin:changeProcessStatus"

pub.prt.tn:handleBizDoc

WmPRT. Sends a Trading Networks BizDocEnvelope (Trading Networks document) to the Process Engine, to allow the document to be processed as part of a business process.

Input Parameters

bizdoc	**Object**com.wm.app.tn.doc.BizDocEnvelope — Trading Networks BizDocEnvelope document that you want to send to the Process Engine.
ConversationID	**String** Optional. Conversation ID for the document. Specify *ConversationID* if the document has no conversation ID or if you want to use an alternate conversation ID.
ProcessModelID	**String** Optional. ID of the process model that you want the Process Engine to use to process the document.
ProcessModelVersion	**String** Optional. Version of the process model that you want the Process Engine to use to process the document.

Note: Specifying *ProcessModelID* and *ProcessModelVersion* overrides the normal process of matching a document to a process model version.

prtIgnoreDocument	**String** Optional. A flag indicating whether this document is to be ignored or processed by the Process Engine. The following values apply:

- `true` — Ignore the document and *do not* send it to the Process Engine for processing. Setting the flag to `true` causes this service to do nothing.
- `false` — Send the document to the Process Engine for processing.

Output Parameters

None.

Usage Notes

Trading Networks automatically sends documents to the Process Engine if it extracts a conversation ID from the document. If you did not have Trading Networks extract a conversation ID, you can use this service to supply a conversation ID and send the document to the Process Engine to be processed as part of a business process.

pub.prt.tn:mapCIDtoPID

WmPRT. Sets up a mapping between the specified conversation ID and process instance ID.

This service is a wrapper around pub.prt.correlate:establishCorrelation.

Input Parameters

ConversationID	**String** Conversation ID that you want to map to the specified process instance ID.
ProcessInstanceID	**String** Process instance ID that you want to map to the specified conversation ID.

Output Parameters

success	**String** Flag indicating whether the mapping was established. The following values apply: - `true` — The mapping was established. - `false` — The mapping was not established.

Note: If this service runs to completion, the mapping has been established.

Usage Notes

The Process Engine automatically establishes this mapping when a Trading Networks document (bizdoc) is used to start a process or is modeled as an output from a process step. Use this service when there is no way for the Process Engine to determine the mapping itself (for example, when a process is started with a non-Trading Networks document and later waits for a Trading Networks document).

Use this service with care. Be sure to create correct conversation ID to process instance ID mappings; an invalid mapping could prevent other processes from completing successfully.

See Also

on page 83"pub.prt.tn:getPIDforCID"

pub.prt.tn:MatchBizDoc

WmPRT. Matches the supplied business document ID to a valid process model. This service provides support for webMethods Rosetta Net in Process Engine.

Input Parameters

bizdoc	**Object** com.wm.app.tn.doc.BizDocEnvelope — Trading Networks BizDocEnvelope document that you want to send to the Process Engine.
useThisMid	**String** Contains the model identifier to match to the *bizdoc* .
requireStart	**String** A flag that indicates whether the associated step starts the business process. The following values apply: ■ `true` — Specify that the step associated with the specified *bizdoc* is a step that starts the business process. ■ `false` — Specify that the step associated with the *bizdoc* does not have to start the business process.

Output Parameters

modelId	**String** The process model identifier that matches the specified business document.
trackCount	**String** The internal process track count for the matching process instance.

stepID	**String** The step identifier.
errorMessage	**String** Text of any generated error message (present only if a Service Exception is raised).

Usage Notes

The service invokes the original TNDispatcher.matchBizDoc() that returns the MatchResult object. This object is parsed for the output parameters used by the Rosetta Net implementation.

pub.prt.tn:RoleInfo

WmPRT. Document type that describes information maintained for roles in a process.

Parameters

ProfileID	**String**Trading Networks internal ID of this trading partner.
CorporationName	**String** Corporation name that is specified in the Trading Networks profile for this trading partner.
OrgUnitName	**String** Organizational unit name that is specified in the Trading Networks profile for this trading partner.
Type	**String** The type of software the partner uses to connect to the trading network. The following values apply: ■ `TNServer` — The partner is using webMethods Trading Networks. ■ `TNPartner` — The partner is using webMethods for Partners. ■ `Browser` — The partner is using a Web browser. ■ `Other` — The partner is using some other method.
Status	**String** Status (active or inactive) of the Trading Networks profile for this trading partner.
PreferredProtocol	**String** The delivery protocol that the partner prefers you to use when sending documents to it. The following values apply: ■ `ftp1` - The partner prefers documents sent using the primary FTP protocol.

- `ftp2` - The partner prefers documents sent using the secondary FTP protocol.
- `http1` - The partner prefers documents sent using the primary HTTP protocol.
- `http2` - The partner prefers documents sent using the secondary HTTP protocol.
- `https1` - The partner prefers documents sent using the primary HTTPS protocol.
- `https2` - The partner prefers documents sent using the secondary HTTPS protocol.
- `smtp1` - The partner prefers documents sent using the primary e-mail protocol.
- `smtp2` - The partner prefers documents sent using the secondary e-mail protocol.
- `null` - The partner prefers documents sent using the polling protocol.

LastSendingLocale **String** Locale associated with the last transmission from this trading partner. Whenever a document is received from this trading partner, this field is updated with the locale information specified in the transmission. For example, if the trading partner uses HTTP to post an XML document to a Trading Networks server and specifies the "ja_JA" locale in the HTTP transmission, this field will contain `ja_JA`.

Index

Q

R

S

T

U

V

W

X

Z

Symbols