



Monitoring WebSphere Process Server



Monitoring WebSphere Process Server

Note

Before using this information, be sure to read the general information in the Notices section at the end of this document.

31 March 2008

This edition applies to version 6, release 1, modification 0 of WebSphere Process Server for Multiplatforms (product number 5724-L01) and to all subsequent releases and modifications until otherwise indicated in new editions.

To send us your comments about this document, send an e-mail message to doc-comments@us.ibm.com. We look forward to hearing from you.

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

© Copyright International Business Machines Corporation 2006, 2008. All rights reserved.

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Chapter 1. Service component monitoring overview 1

Common Event Infrastructure	1
Common Base Event model	3
Why use monitoring?	4
What do you monitor?	4
How do you enable monitoring?.	6

Chapter 2. Enabling and configuring service component monitoring 9

Monitoring performance	9
Performance Monitoring Infrastructure statistics	9
Application Response Measurement statistics for the Service Component Architecture	14
Monitoring service component events	28
Enabling monitoring of business process and human task events	28
Configuring logging for service component events	29
Monitoring service components with the Common Event Infrastructure server	35
Session monitoring	38

Chapter 3. Viewing monitored events 41

Viewing performance metrics with the Tivoli Performance Viewer	41
Viewing and interpreting service component event log files.	42

Viewing events with the Common Base Event browser	44
Specifying the events to view	44
Working with events returned from the event browser	45
Troubleshooting the Common Base Event browser	46

Chapter 4. Event catalog 47

The Common Base Event standard elements	47
Business objects in events.	48
Business Process Choreographer events	49
Monitoring business process events	49
Monitoring human task events	67
Process server events	75
Resource Adapter events	75
Business rule events	77
Business state machine events	77
Map events	79
Mediation events	79
Recovery events	80
Service Component Architecture events	81
Selector events	81

Notices 83

Chapter 1. Service component monitoring overview

A conceptual overview of the reasons you monitor service components on the process server; which event points within the service components you select to monitor; and, how to configure monitoring on your system.

WebSphere® Process Server provides capabilities for monitoring service components to aid in system administration functions, such as performance tuning and problem determination. It goes beyond these traditional functions by also providing the capability for persons who are not necessarily information technology specialists to continually monitor the processing of the service components within the applications deployed on your system. By overseeing the overall processing flow of the interconnected components, you can ensure that your system is producing what you expect it to produce.

WebSphere Process Server operates on top of an installation of WebSphere Application Server, and, consequently, uses much of the functionality of the application server infrastructure for monitoring system performance and troubleshooting. It also includes some extra functionality that is specifically designed for monitoring process server service components. This section of the WebSphere Process Server Information Center will focus on how you monitor process server-specific service components. It is intended to supplement the monitoring and troubleshooting topics found in the WebSphere Application Server, Version 6.1 Information Center; therefore, you should refer to that documentation for details of the other monitoring capabilities in the combined product.

Common Event Infrastructure

Common Event Infrastructure is an embeddable technology intended to provide basic event management services to applications that require those services.

This event infrastructure serves as an integration point for consolidation and persistence of raw events from multiple, heterogeneous sources, and distribution of those events to event consumers. Events are represented using the Common Base Event model, a standard, XML-based format defining the structure of an event. For more information, see the Common Base Event model sub-topic.

By using this common infrastructure, diverse products that are not tightly coupled with one another can integrate their management of events, providing an end-to-end view of enterprise resources and correlating events across domain boundaries. For example, events generated by a network monitoring application can be correlated with events generated by a security application. Such correlation can be difficult to achieve when each product uses its own approach to event management.

Common Event Infrastructure provides facilities for generation, propagation, persistence, and consumption of events, but it does not define the events themselves. Instead, application developers and administrators define event types, event groups, filtering, and correlation.

Common Event Infrastructure components

Common Event Infrastructure consists of the following major components:

Common Base Event

The Common Base Event component supports the creation of events and access to their property data. Event sources use the Common Base Event APIs to create new events conforming to the Common Base Event model; event consumers use the APIs to read property data from received events. In addition, applications can convert events to and from XML text format, supporting interchange with other tools. The Common Base Event component is part of the Eclipse Test and Performance Tools Platform (TPTP).

Emitter

The emitter component supports the sending of events. After an event source creates an event and populates it with data, the event source submits the event to an emitter. The emitter optionally performs automatic content completion and then validates the event to ensure that it conforms to the Common Base Event specification. It also compares the event to configurable filter criteria. If the event is valid and passes the filter criteria, the emitter sends the event to the event service. An emitter can send events to the event service either synchronously (using Enterprise JavaBeans™ calls) or asynchronously (using a Java™ Message Service queue).

Event service

The event service is the conduit between event sources and event consumers. The event service receives events submitted to emitters by event sources. It stores events in a persistent data store, and then distributes them asynchronously to subscribed event consumers. In addition, the event service supports synchronous queries of historical events from the persistent store.

Event catalog

The event catalog is a repository of event metadata. Applications use the event catalog to retrieve information about classes of events and their permitted content.

In addition, an application or solution using Common Event Infrastructure might also include the following components (which are not part of the infrastructure itself):

Event source

An event source is any application that uses an emitter to send events to the event service.

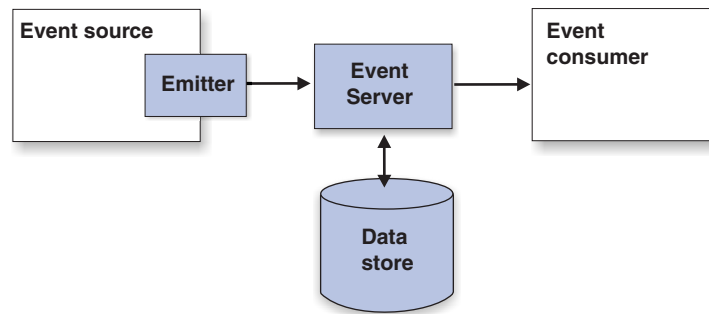
Event consumer

An event consumer is any application that receives events from the event service.

Event catalog application

An event catalog application is any application that stores or retrieves event metadata in the event catalog. This might be a management or development tool; it might also be an event source or event consumer.

The following diagram shows the general flow of events from event source to event consumer using Common Event Infrastructure.



Common Base Event model

The Common Base Event model is a standard that defines a common representation of events that is intended for use by enterprise management and business applications. This standard, developed by the IBM[®] Autonomic Computing Architecture Board, supports encoding of logging, tracing, management, and business events using a common XML-based format, making it possible to correlate different types of events that originate from different applications. The Common Base Event model is part of the IBM Autonomic Computing Toolkit; for more information, see <http://www.ibm.com/autonomic>.

Common Event Infrastructure currently supports version 1.0.1 of the specification.

The basic concept behind the Common Base Event model is the *situation*. A situation can be anything that happens anywhere in the computing infrastructure, such as a server shutdown, a disk-drive failure, or a failed user login. The Common Base Event model defines a set of standard situation types that accommodate most of the situations that might arise (for example, `StartSituation` and `CreateSituation`).

An *event* is a structured notification that reports information related to a situation. An event reports three kinds of information:

- The situation itself (what has happened)
- The identity of the affected component (for example, the server that has shut down)
- The identity of the component that is reporting the situation (which might be the same as the affected component)

The Common Base Event specification defines an event as an XML element containing properties that provide all three kinds of information. These properties are encoded as attributes and subelements of the root element, `CommonBaseEvent`.

The Common Base Event format is extensible. In addition to the standard event properties, an event can also contain extended data elements, which are application-specific elements that can contain any kind of information relevant to the situation. The *extensionName* attribute labels an event with an optional classification name (an event class), which indicates to applications what sort of extended data elements to expect. The event catalog stores event definitions that describe these event classes and their allowed content.

For complete details on the Common Base Event format, see the specification document and XSD schema included in the IBM Autonomic Computing Toolkit.

Why use monitoring?

You monitor service components within WebSphere Process Server to assess performance, to troubleshoot problems, and evaluate the overall processing progress of service components that make up the applications deployed on your system.

Service components are the integral functions incorporated into WebSphere Process Server, with which you can easily create and deploy applications on your system that mirror the processes employed in your enterprise. Effectively monitoring those service components is, therefore, essential to managing the tasks which the process server is intended to accomplish. There are three main reasons you need to monitor service components on the process server:

Problem determination

You can diagnose particular errors by using the logging and tracing facilities provided by WebSphere Application Server, which underlies WebSphere Process Server. For example, if a particular application is not producing the expected results, you can set up a logger to monitor the processing of the service components that comprise that application. You can have the log output published to a file, which you can then examine to pinpoint the cause of the problem. Troubleshooting is a task that is of importance to system administrators and others concerned with the maintenance of system hardware and software.

Performance tuning

You can monitor certain performance statistics that most process server-specific service components produce. Use this information to maintain and tune your system health, and ensure that your applications are tuned optimally and efficiently. You can also spot situations where one or more of your services are performing at a poor level, which may indicate that other problems are present in your system. Like problem determination, performance tuning is a task typically performed by information technology specialists.

Assessing the processing of service components

Problem determination and performance tuning are tasks you perform on a short-term basis, to solve a particular issue or problem. You can also set up the process server to continually monitor the service components incorporated into the applications deployed on your system. This type of service component monitoring is of importance to those who are responsible for designing, implementing, and ensuring that the processes achieve their design goals, and may be accomplished persons who are not necessarily specialists in information technology.

What do you monitor?

You can monitor service component events in WebSphere Process Server by selecting certain points that a service component event reaches during processing. Each service component defines these event points, which "fire" an event when the application processes at that given point. You can also monitor performance statistics for service component events.

Regardless of the type of monitoring you intend to perform on your service components (problem determination, performance tuning, or process monitoring), you will actually be monitoring a certain point that is reached during these components processing. This point is referred to as an *event point*, and it is these

points that you select to be monitored. Each event point encapsulates the service component kind tag, an optional *element* kind (which are specific functions of a service component type), and the *nature* of the event. All of these factors will determine the type of event fired by monitoring.

Event natures describe the situations required to generate (or "fire") events during the processing of service components. These natures are essentially key points in the logic structure of a service component that you select to be monitored. The most common natures for service component events are ENTRY, EXIT, and FAILURE, but there are many other natures depending on the particular component and element. Whenever an application containing the specified service component is subsequently invoked, an event is fired every time the processing of a service component crosses the points corresponding to the event nature.

As an example of how events are defined for a service component kind, the MAP service component kind can directly fire events with natures of ENTRY, EXIT, and FAILURE. It also includes an element kind, called Transformation, which defines a specific type of functionality within the MAP component kind. This element also fires events with ENTRY, EXIT, and FAILURE natures. Consequently, the MAP service component kind can fire up to six different events depending on the combination of elements and natures that you specify. The list of all service components, their elements, and their event natures is contained in the event catalog.

Monitoring is a separate layer of functionality that lies atop the processing of your applications, and does not interfere with the processing of your service components. Monitoring is concerned with service component processing only insofar as it detects activity at a specified event point. When this happens, an event is fired by monitoring, which determines where the event is sent, and what data is contained in that event, based on the type of monitoring you are performing, as detailed below:

Performance metrics

If you are monitoring a service component in order to gather performance metrics, light weight events are fired to the Performance Monitoring Infrastructure. You can select for monitoring one or more of the three performance statistics generated for process server-specific server components:

- A counter for each EXIT event nature – this counts successful computations.
- A counter for each FAILURE event nature – this counts failed computations
- The processing duration calculated between corresponding ENTRY and EXIT events (synchronous computations only).

You can also monitor the performance of applications at the Service Component Architecture (SCA) level by using Application Response Measurement (ARM) statistics. These measures allow you to monitor an application at a much finer level of detail within the application than is otherwise available in other service component events. You can use these statistics to monitor many different points between initial application call invocations and service responses, when they use the SCA.

Service component events with business objects

If you want to capture the data from events fired by monitoring at specified event points in service component, then you would configure the process server to generate the event and its data to be encoded in Common

Base Event formats. You can specify the level of detail of business object data to capture in each service component event. You can publish these events to either a logger or to the Common Event Infrastructure (CEI) bus, which directs the output to a specially configured CEI server database.

How do you enable monitoring?

There are several methods that you can use to specify service component event points for monitoring, depending on the type of monitoring you are planning to do on the process server.

Performance statistics

For Performance Monitoring Infrastructure (PMI) statistics, use the administrative console to specify the particular event points and their associated performance measurements that you want to monitor. After you start monitoring service component performance, the generated statistics are published at certain intervals to the Tivoli® Performance Viewer. You can use this viewer to watch the results as they occur on your system, and, optionally, log the results to a file that can be later viewed and analyzed within the same viewer.

For Application Response Measurement (ARM) statistics, use the administrative console Request Metrics section to specify and the statistics you want to monitor. See the WebSphere Application Server, Version 6.1 Information Center for details and instructions on how to work with request metrics.

Common Base Events for problem determination and business process monitoring

You can specify, at the time you create an application, to monitor service component event points — along with a certain level of detail for those events — on a continual basis after the application is deployed on a running server. You can also select event points to monitor after the application has been deployed and the events invoked at least once on the process server. In both cases, the events generated by monitoring will be fired across the Common Event Infrastructure (CEI) bus. These events can be published to a log file, or to a configured CEI Server database. WebSphere Process Server supports two types of Common Base Event enablement for problem determination and business process monitoring:

Static Certain event points within an application and their level of detail can be tagged for monitoring using WebSphere Integration Developer tooling. The selections indicate what event points are to be continuously monitored, and are stored in a file with a .mon extension that is distributed and deployed along with the process server application. Once the process server is configured to use a CEI server, the monitoring function will begin firing service component events to a CEI server whenever the specified services are invoked. As long as the application is deployed on the process server, the service component event points specified in the .mon file will be constantly monitored until the application is stopped. You can specify additional events to be monitored in a running application, and increase the detail level for event points that are already monitored. But as long as that application remains active you cannot stop, or lower the detail level of, the monitored event points specified by the .mon of the deployed application.

Dynamic

If additional event points need to be monitored during the processing of an application without shutting down the server, then you can use dynamic monitoring. Use the administrative console to specify service component event points for monitoring, and set detail level for the payload that will be included in the Common Base Event. A list is compiled of the event points that have been reached by a processed service component after the process server was started. Choose from this list individual event points or groups of event points for monitoring, with the service component events directed either to the logger or to the CEI server database.

The primary purpose of the Dynamic enablement is for creating correlated service component events that are published to logs, which allow you to perform problem determination on services. Service component events can be large — depending on how much data is being requested — and can tax database resources if you choose to send events to the CEI server. Consequently, you should publish dynamically monitored events to the CEI server only if you need to read the business data of the events, or if you otherwise need to keep a database record of the events. If, however, you are monitoring a particular session, then you will need to use the CEI server database to access the service component events related to that session.

Chapter 2. Enabling and configuring service component monitoring

In order to monitor the service components on the process server, you must first enable the monitoring capabilities. Then you must specify the events you want to monitor, the information you want to capture from the event, and the method used to publish the results.

Monitoring performance

Performance measurements are available for service component event points, and are processed through the Performance Monitoring Infrastructure. You configure the process server to gather performance metrics from service component event points. You can also collect Service Component Architecture-specific performance statistics directly from service invocations of applications.

Whether you are tuning WebSphere Process Server service components for optimal efficiency or diagnosing a poor performance, it is important to understand how the various run time and application resources are behaving from a performance perspective. The Performance Monitoring Infrastructure (PMI) provides a comprehensive set of data that explains the runtime and application resource behavior. Using PMI data, the performance bottlenecks in the application server can be identified and fixed. PMI data can also be used to monitor the health of the application server.

The PMI is included in the base WebSphere Application Server installation. This section provides only supplemental information about performance monitoring as it relates to the service components specific to WebSphere Process Server; therefore, consult the information in the WebSphere Application Server documentation for using PMI with other parts of the entire product.

The service component event points specific to WebSphere Process Server that can be monitored by the PMI are those that typically have ENTRY, EXIT, and FAILURE event natures. Event sources which are not defined according to this pattern are not supported. Events that are supported have three types of performance statistics that can be measured:

- Successful invocations.
- Failed invocations.
- Elapsed time for event completion.

You can also monitor performance statistics derived from the service invocations of applications by using the Application Response Measurement (ARM) statistics. These statistics measure the actual runtime processes that underlie the process server service component events comprising an enterprise application. You can derive a variety of performance measurements for the processing of your applications using these statistics.

Performance Monitoring Infrastructure statistics

You can monitor three types of performance statistics using the Performance Monitoring Infrastructure: the number of successful invocations, the number of

failures, and the elapsed time to completion of an event. These statistics are only available for events that have event natures of type ENTRY, EXIT, and FAILURE.

Enabling PMI using the administrative console

To monitor performance data you must first enable the Performance Monitoring Infrastructure on the server.

About this task

You can enable the Performance Monitoring Infrastructure (PMI) through the administrative console.

Procedure

1. Open the administrative console.
2. Click **Servers > Application Servers** in the console navigation tree.
3. Click *server_name*.

Note: From the administrative console, you can click **Monitoring and Tuning > Performance Monitoring Infrastructure (PMI) > server_name** to open the same panel

4. Click the **Configuration** tab.
5. Select the **Enable Performance Monitoring Infrastructure (PMI)** check box.
6. Optional: Select the check box for **Use sequential counter updates** to enable precise statistic update.
7. Go back to the server PMI configuration page by clicking the server name link.
8. Click **Apply** or **OK**.
9. Click **Save**.
10. Restart the server.

What to do next

The changes you make will not take effect until you restart the server.

Event performance statistics

Performance monitoring statistics are available for most server events. You can monitor the counts of invocation requests (good and bad) and the time it took to complete the event.

You can use the Performance Monitoring Infrastructure (PMI) to monitor three performance statistics generated by certain server events, as shown in the following table:

Table 1. PMI statistics for events

Statistic name	Type	Description
BadRequests	Counter	Number of failed invocations of the event.
GoodRequests	Counter	Number of successful invocations of the event.
ResponseTime	Timer	Elapsed time for event completion.

. These statistics are limited to service component events with elements having ENTRY, EXIT, and FAILURE natures. Each statistic is created for a single event of a given server event type in an application. All performance measurements are either *counters* (a cumulative number of the firings of a given event point), or *timers* (the duration, measured in milliseconds, between the firings of two event points). Each event kind (and their relevant elements) that can be monitored are listed below:

Table 2. Event types and elements that can produce event performance statistics

Event type	Element(s)
Business process	Process Invoke Staff Receive Wait Compensate Pick Scope
Human task	Task
Business rule	Operation
Business state machine	Transition Guard Action EntryAction ExitAction
Selector	Operation
Map	Map Transformation
Mediation	OperationBinding ParameterMediation
Resource adapter	InboundEventRetrieval InboundEventDelivery Outbound

Specifying performance statistics to monitor

You can specify single statistics, multiple statistics, or groups of related statistics for monitoring through the Performance Monitoring Infrastructure by using the administrative console.

Before you begin

Ensure that you have enabled performance monitoring, and that you have at least once invoked the event you want to monitor before performing this task.

Procedure

1. Open the administrative console.
2. Select **Monitoring and Tuning > Performance Monitoring Infrastructure**.
3. Select the server or node agent that contains the event points that you want to monitor.

Note: You cannot choose to monitor statistics on a cluster; you can only do so on a specific server or node.

4. Expand some of the groups, such as **WBISStats.RootGroup** or **Enterprise Beans**. All of the statistics that can be monitored are within the listed groups. Some statistics may not be listed because they have not been invoked since the process server was last started.
5. Select a statistic you want to monitor from within the tree on the left side of the panel, and then select the statistics that you want to collect on the right side and click **Enable**. Repeat this for all statistics that you want to monitor.
6. Go back to the server PMI configuration page by clicking the server name link.
7. Click **Apply** or **OK**.
8. Click **Save**.

Results

You can now start monitoring the performance of your chosen statistics in the Tivoli Performance Viewer.

Note: When viewing these statistics, you should not mix counter-type statistics with duration-type statistics. Counters are cumulative, and the scales against which they are graphed them can quickly grow depending on your application. Duration statistics, in contrast, tend to remain within a certain range because they represent the average amount of time that it takes your system to process each event. Consequently, the disparity between the statistics and their relative scales may cause one or the other type of statistic to appear skewed in the viewer graph.

Tutorial: Service component performance monitoring

This tutorial will guide you through an example of how to set up service component event monitors that are published to the Performance Monitoring Infrastructure (PMI), and how to view the resulting performance statistics on the Tivoli Performance Viewer (TPV). This exercise will demonstrate how performance monitoring of service component event points differs from monitoring using the Common Event Infrastructure (CEI) server and loggers. The major difference that you will notice is that you select an entire service component element for performance monitoring, instead of individual events with specific natures. Because WebSphere Process Server can monitor performance only on service component elements having events with ENTRY, EXIT, and FAILURE natures, you will have only those kinds of service component elements available to you to select for monitoring.

While the service component event points ENTRY, EXIT, and FAILURE are identical for all monitoring types, the process server performance monitoring function fires "minimized" events that do not contain all of the information encompassed in CEI events. These events are sent to the PMI, which calculates these performance statistics from corresponding sets of events:

- Successful invocation — the firing of an event of nature type EXIT that follows a corresponding ENTRY event.
- Failed invocation — the firing of an event with a FAILURE nature following a corresponding ENTRY event.
- Time for successful completion — the elapsed time between the firing an ENTRY event and the firing of the corresponding EXIT event point.

The PMI publishes the statistics to the TPV, which presents cumulative counters for the number of successful and failed invocations and a running average of the completion response times.

Objectives of this tutorial

After completing this tutorial you will be able to:

- Select the performance statistics of service component elements that you want to monitor.
- View and interpret the resulting performance statistics.

Time required to complete this tutorial

This tutorial requires approximately 15-20 minutes to complete.

Prerequisites

In order to perform this tutorial, you must have:

- Configured and started a process server.
- Enabled the PMI on the process server.
- Installed and started the Samples Gallery application on the process server.
- Installed and started the business rules sample application on the process server. Follow the instructions on the Samples Gallery page to set up and run the business rules sample application.

After all of these prerequisites have been completed, run the business rules sample application from the Samples Gallery at least once before proceeding with the tutorial.

Example: Monitoring service component performance: About this task

You will use the business rules sample application for this scenario, where you will monitor all three of the performance statistics: successes, failures, and response times. You should have the web page containing this application already open; keep it open, since you will be running the sample several times after you begin monitoring. Ensure that you have already run the sample at least once, which will cause it to appear in the list of functions that you can select to monitor.

Procedure

1. Open the administrative console.
2. To monitor a cluster, click **Servers > Clusters > *cluster_name*** in the console navigation tree. To monitor a single process server, click **Servers > Application Servers > *server_name*** in the console navigation tree.
3. Click the **Runtime** tab.
4. Click **Performance Monitoring Infrastructure** under Performance.
5. Select **Custom**.
6. Expand **WBISStats.RootGroup > BR > brsample_module.DiscountRuleGroup > Operation**, and select **_calculateDiscount**.
7. Select the check boxes next to **BadRequests**, **GoodRequests**, and **ResponseTime**, and press **Enable**.
8. Click **Monitoring and Tuning > Performance Viewer > Current Activity** in the administrative console navigation tree.
9. Select the check box next to *server_name* and press **Start Monitoring**.
10. Click *server_name*.
11. Expand **WBISStats.RootGroup > BR > brsample_module.DiscountRuleGroup > Operation**, and select the check box next to **_calculateDiscount**.

Results

You should now see a blank graph, and underneath that the names and values for the three statistics. Select the check boxes next to the statistic names, if they are not already checked. The PMI is now ready to publish performance data for the selected event, and the TPV is ready to present the results.

Run the business rules sample application several times, and then watch the performance viewer as it periodically refreshes. Notice that there are now lines on the graph, representing the cumulative number of successful requests and the average response time for each successful request. You can also see the values next to the name for each statistic below the graph. The line for the number of successes should continue to rise as you perform additional invocations of the sample, while the response time line should level off after a few refreshes.

Once you have completed this task, you should understand how WebSphere Process Server implements performance monitoring of service components. You will know how to select service components for monitoring, and how the performance statistics are calculated. You will also be able to start the performance monitors, and view the performance measurements for your applications as they are being used.

Performance monitoring can tax system resources; therefore, after you have completed this task you should stop the monitors. To do this, simply click on the Tivoli Performance Viewer link, select both the node and the server, and press **Stop Monitoring**.

Application Response Measurement statistics for the Service Component Architecture

There are 25 performance statistics that you can monitor at the Service Component Architecture level. You can use these Application Response Measurement statistics, which are either counters or timers, to measure invocations to and responses from services in a variety of patterns.

The Application Response Measurement (ARM) statistics shown in the following tables are — in a very simplified manner — time and count measurements of caller invocations to the Service Component Architecture (SCA) layer, and the results returned from a service. There are, in fact, a number of service invocation patterns that vary between synchronous and asynchronous implementations of deferred responses, results retrievals, callbacks, and one-way invocations. All of these, however, are between the caller invocation and a service, the response from the service, or, in some cases, a data source, with the SCA layer interposed in between.

You can specify the ARM statistics that you want to monitor by opening the **Monitoring and Tuning > Request Metrics** panel on the administrative console. Request metrics information might be either saved to the log file for later retrieval and analysis, be sent to ARM agents, or both. WebSphere Process Server does not ship an ARM agent; however, it supports the use of agents adhering to ARM 4.0. You can choose your own ARM implementation provider to obtain the ARM implementation libraries. Follow the instructions from the ARM provider, and ensure that the ARM API Java archive (JAR) files found in the ARM provider are on the class path so that the WebSphere Process Server can load the needed classes. Then you need to add the following entries into the system properties for

each server by selecting from the administrative console **Application servers > server_name > Process Definition > Java Virtual Machine > Custom Properties** before restarting the server(s):

- `Arm40.ArmMetricFactory` — the full Java class name of your ARM implementation provider’s metrics factory.
- `Arm40.ArmTranReportFactory` — the full Java class name of your ARM implementation provider’s transaction report factory.
- `Arm40.ArmTransactionFactory` — the full Java class name of your ARM implementation provider’s transaction factory .

See the WebSphere Application Server documentation for further details on how to configure the server to collect ARM statistics.

Table 3. Event types and elements that can produce ARM statistics

Event type	Element(s)
Business process	Process
Human task	Task
Business rule	Operation
Business state machine	Transition Guard Action EntryAction ExitAction
Selector	Operation
Map	Map Transformation
Mediation	OperationBinding ParameterMediation
Resource adapter	InboundEventRetrieval InboundEventDelivery Outbound

Table 4. Common. These statistics are common to all service invocation patterns.

Statistic name	Type	Description
GoodRequests	Counter	Number of server invocations not raising exceptions.
BadRequests	Counter	Number of server invocations raising exceptions.
ResponseTime	Timer	Duration measured on the server side between the reception of a request and computing the result.
TotalResponseTime	Timer	Duration measured on the caller side, from the time a caller requests a service to the time when the result is available for the caller. Does not include the processing of the result by the caller.

Table 4. Common (continued). These statistics are common to all service invocation patterns.

Statistic name	Type	Description
RequestDeliveryTime	Timer	Duration measured on the caller side, from the time a caller requests a service to the time when the request is handed over to the implementation on the server side. In a distributed environment the quality of this measurement depends on the quality of synchronization of system clocks.
ResponseDeliveryTime	Timer	The time required to make the result available to the client. In the case of deferred response this time doesn't include the result retrieve time. In a distributed environment the quality of this measurement depends on the quality of synchronization of system clocks.

Table 5. Reference. These statistics occur when a caller makes an invocation to the SCA layer or a data source, without a response from the service.

Statistic name	Type	Description
GoodRefRequests	Counter	Number of caller invocations to the SCA layer that do not raise exceptions.
BadRefRequests	Counter	Number of caller invocations to the SCA layer that do raise exceptions.
RefResponseTime	Timer	Duration measured on the caller side, from the time the caller makes a request to the SCA layer and the time when the results of that call are returned to the caller.
BadRetrieveResult	Counter	Number of caller invocations to a data source that do raise exceptions.
GoodRetrieveResult	Counter	Number of caller invocations to a data source that do not raise exceptions.
RetrieveResultResponseTime	Timer	Duration measured on the caller side, from the time the caller makes a request to the data source and the time when the data source response is returned to the caller.
RetrieveResultWaitTime	Timer	Duration measured on the caller side if a timeout occurs.

Table 6. Target. These statistics occur when there are requests that originate between the service and the SCA or a data source.

Statistic name	Type	Description
GoodTargetSubmit	Counter	Number of SCA invocations to the service that do not raise exceptions.
BadTargetSubmit	Counter	Number of SCA invocations to the service that do raise exceptions.
TargetSubmitTime	Timer	Duration measured on the server side, from the time the SCA makes a request to the service and the time when the results of that call are returned to the SCA.

Table 6. Target (continued). These statistics occur when there are requests that originate between the service and the SCA or a data source.

Statistic name	Type	Description
GoodResultSubmit	Counter	Number of service invocations to the data source that do not raise exceptions.
BadResultSubmit	Counter	Number of service invocations to the data source that do raise exceptions.
ResultSubmitTime	Timer	Duration measured on the server side, from the time the service makes a request to the data source and the time when the results of are returned to the service.

Table 7. Callback. These statistics occur when a callback (a "sibling" of the original call) is present on the caller.

Statistic name	Type	Description
GoodCB	Counter	Number of SCA invocations to the callback that do not raise exceptions.
BadCB	Counter	Number of SCA invocations to the callback that do raise exceptions.
CBTime	Timer	Duration from the time the SCA makes a request to the callback, and the time when the results from the callback are returned to the SCA.
GoodCBSubmit	Counter	Number of invocations from the service to the SCA handling the callback that do not raise exceptions.
BadCBSubmit	Counter	Number of invocations from the service to the SCA handling the callback that do raise exceptions.
CBSubmitTime	Timer	Duration from the time the service makes a request to the SCA handling the callback, and the time when the results from the SCA to the service.



The following topics show how these statistics are used in a variety of implementations:

Synchronous invocations

The ARM performance statistics that can be obtained from a simple SCA call to a service and the response from the service are described.

Parameters

Event monitoring for SCA components includes the event points that are shown in

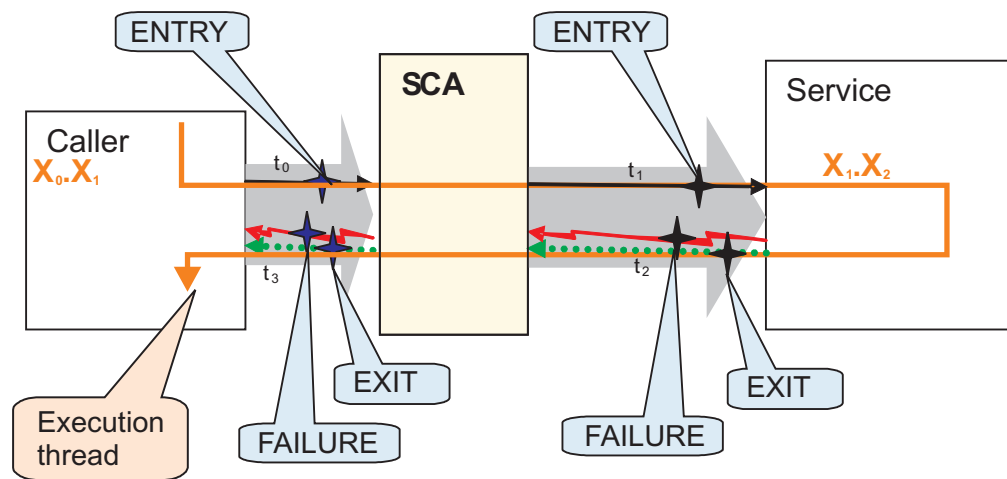
black  , while the event points shown in blue  are used only to calculate and fire PMI/ARM statistics.

In the table and diagram below, the "current" ARM transaction (denoted as X_i) is created when the calling service component was invoked for the first time. If the caller is not a service component, the current arm transaction will be used, or a new will be created. If it is not the starting transaction it will have a parent. This is represented in the following table and diagram with the notation $X_n.X_{n+1}$. These are used to show the transaction lineage. Every SCA invocation starts a new

transaction, which is parented by the current transaction of the caller. You can create new transactions and you can access the current transaction, but this will not modify the SCA transaction lineage.

Table 8. Arm statistics for synchronous invocations of SCA

Statistics	Formula	ARM Transaction
TotalResponseTime	$t_3 - t_0$	$X_0 . X_1$
RequestDeliveryTime	$t_1 - t_0$	$X_1 . X_2$
ResponseDeliveryTime	$t_3 - t_2$	
GoodRequests	$\text{Count}_{\text{EXIT}}$	
BadRequests	$\text{Count}_{\text{FAILURE}}$	
ProcessTime	$t_2 - t_1$	





Deferred response with synchronous implementation

ARM statistics that can be obtained with a synchronous invocation of the request and the returned result sent as output to a data store are shown for a synchronous implementation.

Parameters

Event monitoring for SCA components includes the event points that are shown in

black , while the event points shown in blue  are used only to calculate and fire PMI/ARM statistics.

In the table and diagram below, the "current" ARM transaction (denoted as X_1) is created when the calling service component was invoked for the first time. If the caller is not a service component, the current arm transaction will be used, or a new will be created. If it is not the starting transaction it will have a parent. This is represented in the following table and diagram with the notation $X_n . X_{n+1}$. These are used to show the transaction lineage. Every SCA invocation starts a new transaction, which is parented by the current transaction of the caller. You can create new transactions and you can access the current transaction, but this will not modify the SCA transaction lineage.

Table 9. Invocation of request and return result

Type	Statistics	Formula	ARM Transaction
Common	TotalResponseTime	$t_3 - t_0$	$X_0.X_1$
	RequestDeliveryTime	$t'_0 - t_0$	$X_1.X_2$
	ResponseDeliveryTime	N/A	N/A
	GoodRequests	$Count_{EXIT}$	$X_1.X_2$
	BadRequests	$Count_{FAILURE}$	
	ResponseTime	$t'_1 - t'_0$	
Reference A	GoodRefRequest	$Count_{EXIT}$	$X_1.X_2$
	BadRefRequests	$Count_{FAILURE}$	
	RefResponseTime	$t_1 - t_0$	

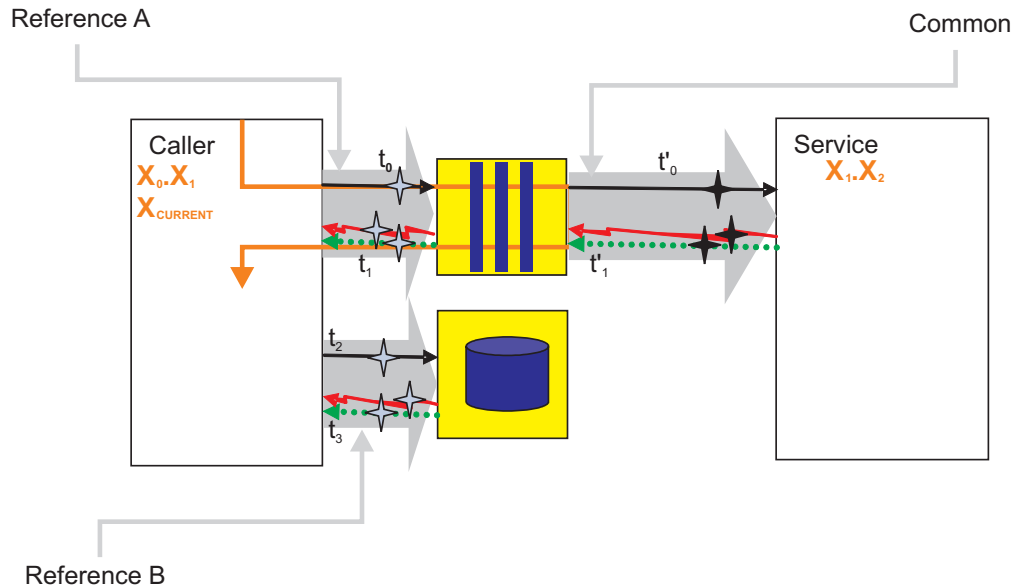


Table 10. Invocation of output to data source



Type	Statistics	Formula	ARM Transaction
Reference B	GoodRetrieveResult	$Count_{EXIT}$	$X_1.X_2$
	BadRetrieveResult	$Count_{FAILURE}$	
	ResultRetrieveResponseTime	$\sum t_3 - t_2$	
	ResultRetrieveWaitTime	$\sum timeout$	

Deferred response with asynchronous implementation

ARM statistics from an asynchronous implementation, where the call to the service and the return result are invoked but the result output is sent to a data store from the service target, are shown.

Parameters

Event monitoring for SCA components includes the event points that are shown in

black , while the event points shown in blue  are used only to calculate and fire PMI/ARM statistics.

In the table and diagram below, the "current" ARM transaction (denoted as X_1) is created when the calling service component was invoked for the first time. If the caller is not a service component, the current arm transaction will be used, or a new will be created. If it is not the starting transaction it will have a parent. This is represented in the following table and diagram with the notation $X_n.X_{n+1}$. These are used to show the transaction lineage. Every SCA invocation starts a new transaction, which is parented by the current transaction of the caller. You can create new transactions and you can access the current transaction, but this will not modify the SCA transaction lineage.

Table 11. Invocation of request and return result

Type	Statistics	Formula	ARM Transaction
Common	TotalResponseTime	$t_3 - t_0$	$X_0.X_1$
	RequestDeliveryTime	$t'_0 - t_0$	$X_1.X_2$
	ResponseDeliveryTime	$t'_{03} - t'_2$	
	GoodRequests	$\text{Count}_{\text{EXIT}}$	
	BadRequests	$\text{Count}_{\text{FAILURE}}$	
	ResponseTime	$t'_3 - t'_0$	
Reference A	GoodRefRequest	$\text{Count}_{\text{EXIT}}$	$X_0.X_1$
	BadRefRequests	$\text{Count}_{\text{FAILURE}}$	
	RefResponseTime	$t_1 - t_0$	
Target A	GoodTargetSubmit	$\text{Count}_{\text{EXIT}}$	$X_1.X_2$
	BadTargetSubmit	$\text{Count}_{\text{FAILURE}}$	
	TargetSubmitTime	$t'_1 - t'_0$	

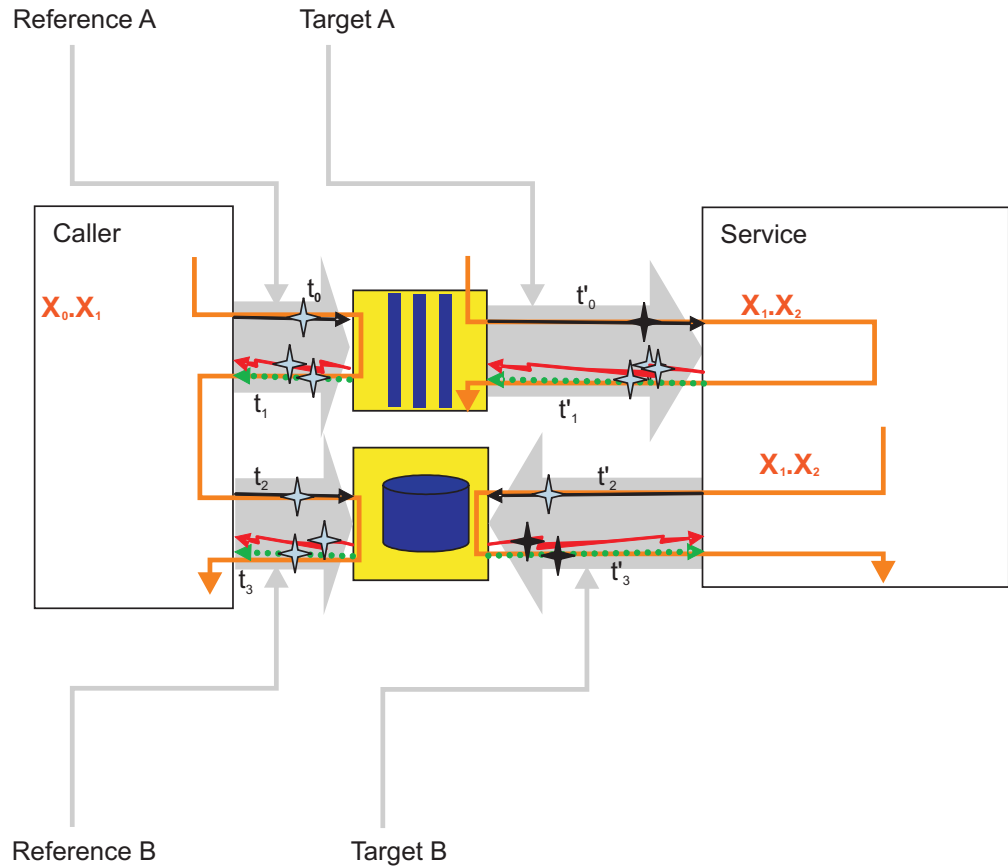


Table 12. Invocation of return result to a data store



Type	Statistics	Formula	ARM Transaction
Reference B	GoodResultSubmit	$\text{Count}_{\text{EXIT}}$	$X_0 \cdot X_1$
	BadResultSubmit	$\text{Count}_{\text{FAILURE}}$	
	ResultResponseTime	$t'_3 - t'_2$	
Target B	GoodResultRetrieve	$\text{Count}_{\text{EXIT}}$	$X_1 \cdot X_2$
	BadResultRetrieve	$\text{Count}_{\text{FAILURE}}$	
	ResultRetrieveResponseTime	$\sum t_3 - t_2$	
	ResultRetrieveWaitTime	$\sum \text{timeout}$	

Deferred response with asynchronous result retrieve

The ResultRetrieve ARM statistic can be correlated to some original request using the ARM transactions only if $X_{\text{PARENT-1}}$ and $X_{\text{PARENT-2}}$ have a common ancestor transaction. The invocation of request, and result retrieve occur on different threads

Parameters

Event monitoring for SCA components includes the event points that are shown in

black  , while the event points shown in blue  are used only to calculate and fire PMI/ARM statistics.

In the table and diagram below, the "current" ARM transaction (denoted as X_1) is created when the calling service component was invoked for the first time. If the caller is not a service component, the current arm transaction will be used, or a new will be created. If it is not the starting transaction it will have a parent. This is represented in the following table and diagram with the notation $X_n.X_{n+1}$. These are used to show the transaction lineage. Every SCA invocation starts a new transaction, which is parented by the current transaction of the caller. You can create new transactions and you can access the current transaction, but this will not modify the SCA transaction lineage.

Table 13. Invocation of request and return result

Type	Statistics	Formula	ARM Transaction
Common	TotalResponseTime	$t_3 - t_0$	$X_0.X_1$
	RequestDeliveryTime	$t'_0 - t_0$	$X_1.X_2$
	ResponseDeliveryTime	N/A	N/A
	GoodRequests	$Count_{EXIT}$	$X_1.X_2$
	BadRequests	$Count_{FAILURE}$	
	ResponseTime	See specific diagrams	
Reference A	GoodReferenceRequest	$Count_{EXIT}$	$X_1.X_2$
	BadReferenceRequests	$Count_{FAILURE}$	
	ReferenceResponseTime	$t_1 - t_0$	

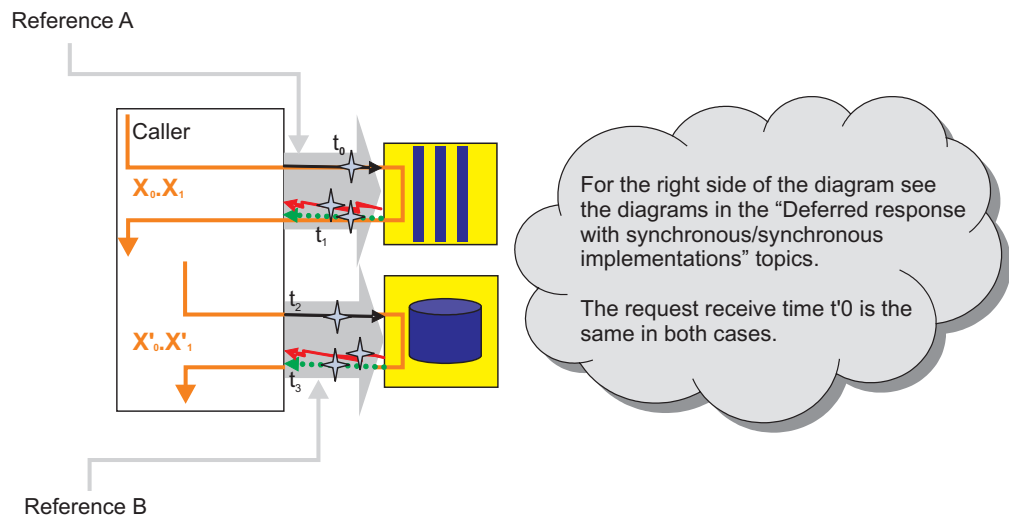


Table 14. Invocation of request and return result



Type	Statistics	Formula	ARM Transaction
Reference B	GoodRetrieveResult	$Count_{EXIT}$	$X'_0.X'_1$
	BadRetrieveResult	$Count_{FAILURE}$	
	RetrieveResultResponseTime	$\sum t_3 - t_2$	
	RetrieveResultWaitTime	$\sum \text{timeout}$	

Asynchronous callback with synchronous implementation

ARM statistics available when callback requests and callback executions use different threads on a synchronous implementation.

Parameters

Event monitoring for SCA components includes the event points that are shown in

black  , while the event points shown in blue  are used only to calculate and fire PMI/ARM statistics.

In the table and diagram below, the "current" ARM transaction (denoted as X_1) is created when the calling service component was invoked for the first time. If the caller is not a service component, the current arm transaction will be used, or a new will be created. If it is not the starting transaction it will have a parent. This is represented in the following table and diagram with the notation $X_n \cdot X_{n+1}$. These are used to show the transaction lineage. Every SCA invocation starts a new transaction, which is parented by the current transaction of the caller. You can create new transactions and you can access the current transaction, but this will not modify the SCA transaction lineage.

Table 15. Invocation of request and return result

Type	Statistics	Formula	ARM Transaction
Common	TotalResponseTime	$t_2 - t_0$	$X_0 \cdot X_1$
	RequestDeliveryTime	$t'_0 - t_0$	$X_1 \cdot X_2$
	ResponseDeliveryTime	$t_2 - t'_1$	
	GoodRequests	$\text{Count}_{\text{EXIT}}$	
	BadRequests	$\text{Count}_{\text{FAILURE}}$	
	ResponseTime	$t_3 - t_2$	
Reference	GoodRefRequest	$\text{Count}_{\text{EXIT}}$	$X_1 \cdot X_2$
	BadRefRequests	$\text{Count}_{\text{FAILURE}}$	
	RefResponseTime	$t'_1 - t'_0$	

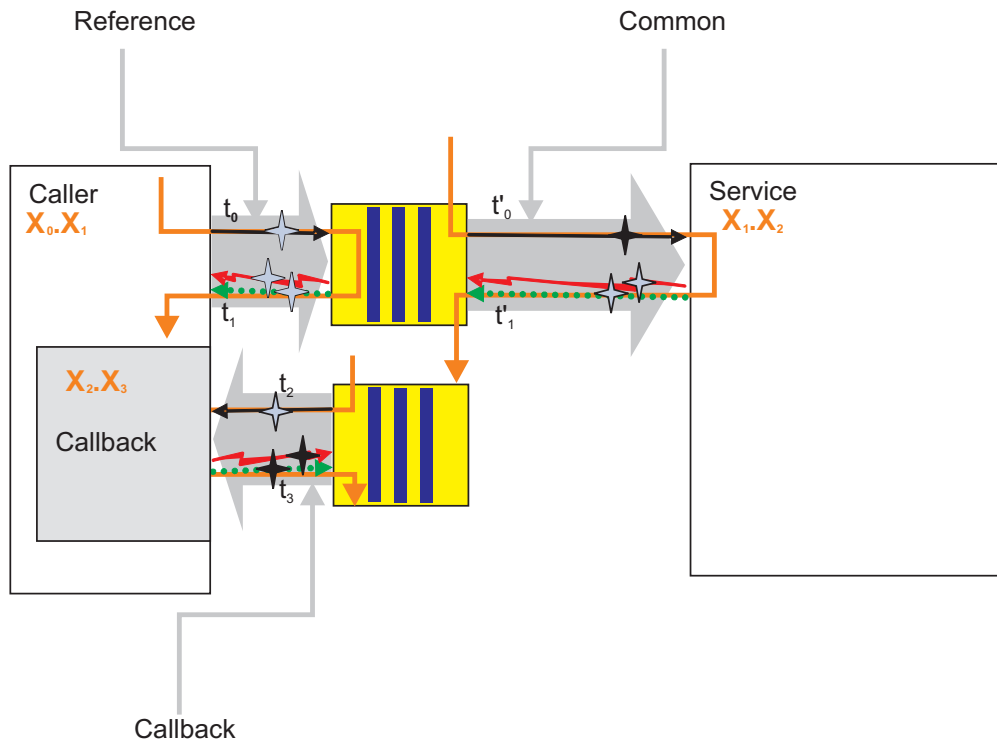


Table 16. Invocation of callback



Type	Statistics	Formula	ARM Transaction
Callback	GoodCB	$\text{Count}_{\text{EXIT}}$	$X_1.X_3$
	BadCB	$\text{Count}_{\text{FAILURE}}$	
	CBTime	$t_3 - t_2$	

Asynchronous callback with asynchronous implementation

ARM statistics available for callback requests and callback executions using different threads with an asynchronous implementation

Parameters

Event monitoring for SCA components includes the event points that are shown in

black , while the event points shown in blue  are used only to calculate and fire PMI/ARM statistics.

In the table and diagram below, the "current" ARM transaction (denoted as X_1) is created when the calling service component was invoked for the first time. If the caller is not a service component, the current arm transaction will be used, or a new will be created. If it is not the starting transaction it will have a parent. This is represented in the following table and diagram with the notation $X_n.X_{n+1}$. These are used to show the transaction lineage. Every SCA invocation starts a new transaction, which is parented by the current transaction of the caller. You can create new transactions and you can access the current transaction, but this will not modify the SCA transaction lineage.

Table 17. Invocation of request and return result

Type	Statistics	Formula	ARM Transaction
Common	TotalResponseTime	$t_2 - t_0$	$X_0 \cdot X_1$
	RequestDeliveryTime	$t'_0 - t_0$	$X_1 \cdot X_2$
	ResponseDeliveryTime	$t_2 - t'_2$	
	GoodRequests	$\text{Count}_{\text{EXIT}}$	
	BadRequests	$\text{Count}_{\text{FAILURE}}$	
	ResponseTime	$t'_3 - t'_0$	
Reference A	GoodRefRequest	$\text{Count}_{\text{EXIT}}$	$X_0 \cdot X_1$
	BadRefRequests	$\text{Count}_{\text{FAILURE}}$	
	RefResponseTime	$t_1 - t_0$	
Target A	GoodTargetSubmit	$\text{Count}_{\text{EXIT}}$	$X_1 \cdot X_2$
	BadTargetSubmit	$\text{Count}_{\text{FAILURE}}$	
	TargetSubmitTime	$t'_1 - t'_0$	

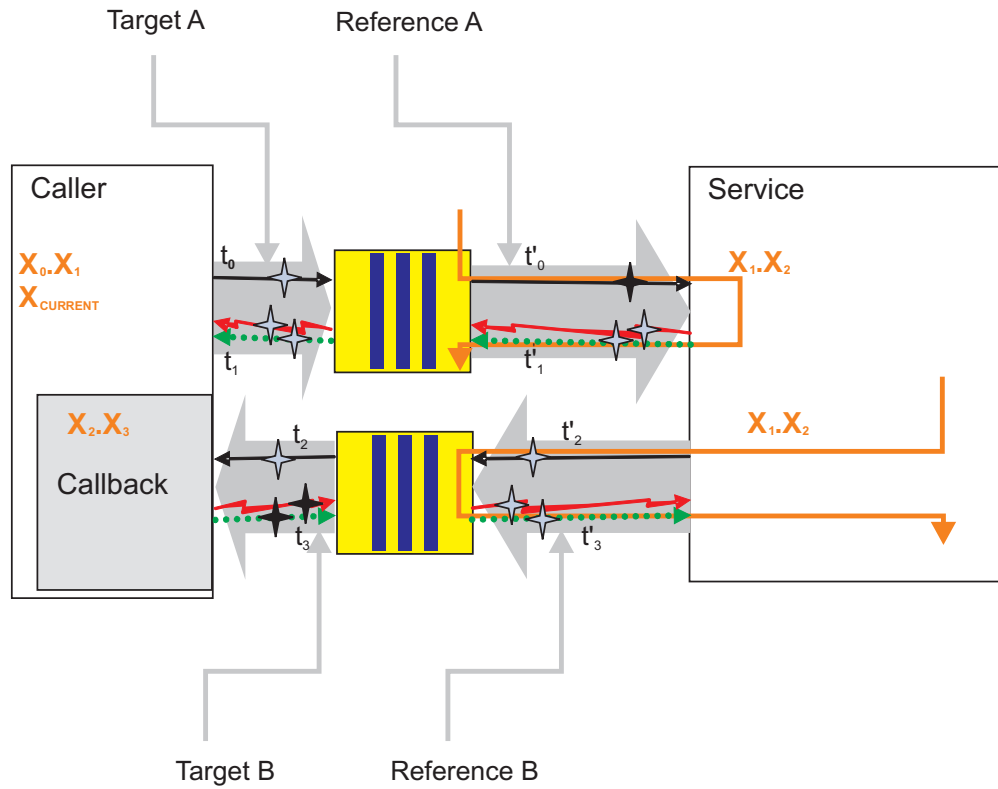


Table 18. Invocation of callback

Type	Statistics	Formula	ARM Transaction
Reference B	GoodCBSubmit	$\text{Count}_{\text{EXIT}}$	$X_1 \cdot X_2$
	BadCBSubmit	$\text{Count}_{\text{FAILURE}}$	
	CBSubmitTime	$t'_3 - t'_2$	

Table 18. Invocation of callback (continued)



Type	Statistics	Formula	ARM Transaction
Target B	GoodCB	Count _{EXIT}	$X_0 \cdot X_1$
	BadCB	Count _{FAILURE}	
	CBTime	$t_3 - t_2$	

Asynchronous one-way with synchronous implementation

ARM statistics when a call is submitted (fire and forget) with a synchronous implementation.

Parameters

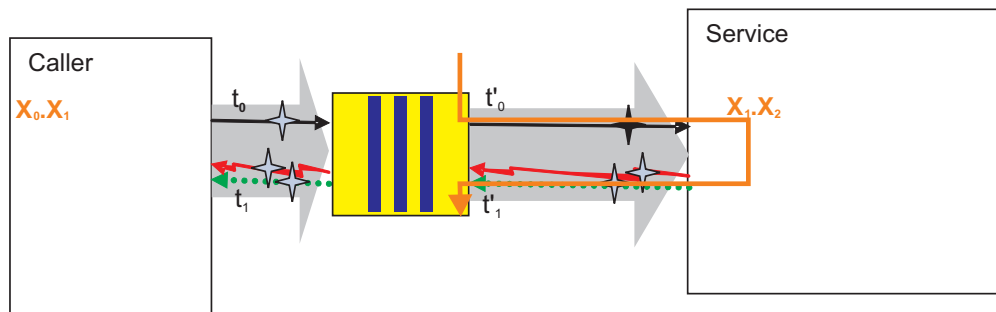
Event monitoring for SCA components includes the event points that are shown in

black , while the event points shown in blue  are used only to calculate and fire PMI/ARM statistics.

In the table and diagram below, the "current" ARM transaction (denoted as X_1) is created when the calling service component was invoked for the first time. If the caller is not a service component, the current arm transaction will be used, or a new will be created. If it is not the starting transaction it will have a parent. This is represented in the following table and diagram with the notation $X_n \cdot X_{n+1}$. These are used to show the transaction lineage. Every SCA invocation starts a new transaction, which is parented by the current transaction of the caller. You can create new transactions and you can access the current transaction, but this will not modify the SCA transaction lineage.

Table 19. Invocation of request and return result

Type	Statistics	Formula	ARM Transaction
Common	TotalResponseTime	$t_1 - t_0$	$X_0 \cdot X_1$
	RequestDeliveryTime	$t'_0 - t_0$	$X_1 \cdot X_2$
	ResponseDeliveryTime	N/A	N/A
	GoodRequests	Count _{EXIT}	$X_1 \cdot X_2$
	BadRequests	Count _{FAILURE}	
	ResponseTime	$t'_1 - t'_0$	





Asynchronous one-way with asynchronous implementation

ARM statistics when a call is submitted (fire and forget) with an asynchronous implementation.

Parameters

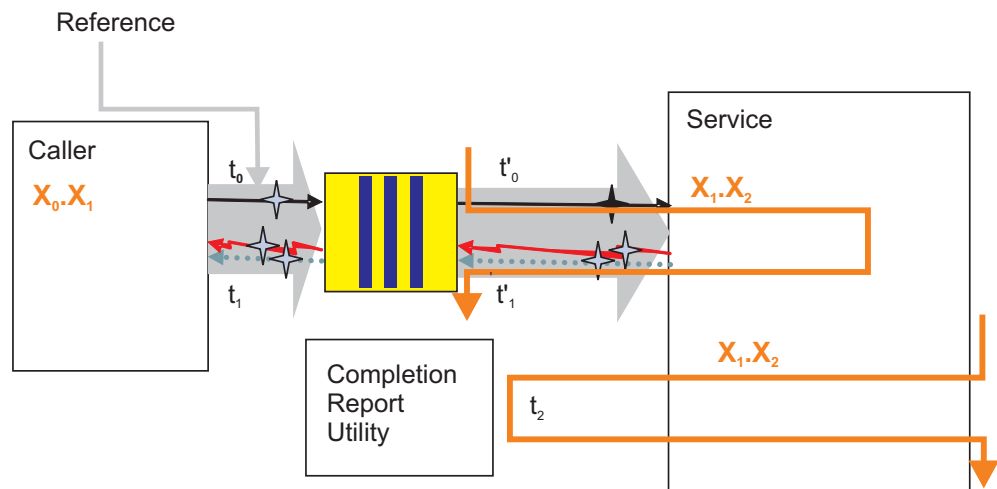
Event monitoring for SCA components includes the event points that are shown in

black , while the event points shown in blue  are used only to calculate and fire PMI/ARM statistics.

In the table and diagram below, the "current" ARM transaction (denoted as X_1) is created when the calling service component was invoked for the first time. If the caller is not a service component, the current arm transaction will be used, or a new will be created. If it is not the starting transaction it will have a parent. This is represented in the following table and diagram with the notation $X_n \cdot X_{n+1}$. These are used to show the transaction lineage. Every SCA invocation starts a new transaction, which is parented by the current transaction of the caller. You can create new transactions and you can access the current transaction, but this will not modify the SCA transaction lineage.

Table 20. Invocation of request and return result

Type	Statistics	Formula	ARM Transaction
Common	TotalResponseTime	$t_1 - t_0$	$X_{\text{PARENT}} \cdot X_{\text{REQUEST}}$
	RequestDeliveryTime	$t'_0 - t_0$	$X_{\text{REQUEST}} \cdot X_{\text{PROCESS}}$
	ResponseDeliveryTime	N/A	N/A
	GoodRequests	$\text{Count}_{\text{EXIT}}$	$X_{\text{REQUEST}} \cdot X_{\text{PROCESS}}$
	BadRequests	$\text{Count}_{\text{FAILURE}}$	
	ResponseTime	$t'_2 - t_0$	
Reference	GoodRefRequest	$\text{Count}_{\text{EXIT}}$	$X_{\text{PARENT}} \cdot X_{\text{REQUEST}}$
	BadRefRequest	$\text{Count}_{\text{FAILURE}}$	
	RefResponseDuration	$t_1 - t_0$	



Monitoring service component events

WebSphere Process Server monitoring can capture the data in a service component at a certain event point. You can view each event in a log file, or you can use the more versatile monitoring capabilities of a Common Event Infrastructure server.

Applications that are deployed on the process server may contain a specification of service component events that will be monitored for as long as the application runs. If you developed the application using the WebSphere Integration Developer, then you can specify service component events to monitor continuously. This specification is included as part of the application, and comes in the form of file with a .mon extension that is read by the process server when the application is deployed. Once the application is started, you will not be able to turn off monitoring of the service components specified in the .mon file. The documentation for the WebSphere Process Server does not address this type of continuous monitoring. For more information about this subject, refer to the WebSphere Integration Developer documentation.

You can use WebSphere Process Server to monitor service component events that are not already specified in the .mon file of the application. You can configure the process server to direct the output of the event monitors to a log file, or to a Common Event Infrastructure server database. The monitored events will be formatted using the Common Base Event standard, but you can regulate the amount of information contained in each event. Use the monitoring facilities in WebSphere Process Server to diagnose problems, analyze the process flow of your applications, or audit how your applications are used.

Enabling monitoring of business process and human task events

You must configure WebSphere Process Server to support monitoring of business process and human task service components before you do any actual monitoring of those service component kinds.

Before you begin

You must have previously created the business process container and the human task container on the process server.

About this task

Perform this task to enable Common Event Infrastructure monitoring support on WebSphere Process Server.

Procedure

1. Open the administrative console.
2. If Business Process Choreographer is configured on a single server, complete the following steps to enable the server to generate business process events:
 - a. In the left frame, expand **Servers** and click **Application servers > *server_name***
 - b. Select **Containers Settings > Business Process Choreographer container settings > Business Process Choreographer container**.
 - c. Click **State Observers**.

- d. Ensure that the boxes for **Audit Logging** and **Common Event Infrastructure Logging** are checked for both the **Business Flow Manager** and the **Human Task Manager**. If the check boxes are not selected, then you must select them and restart the server.
3. If Business Process Choreographer is configured on a cluster, complete the following steps to enable the cluster to generate business process events:
 - a. Select **Clusters > cluster_name**.
 - b. Select **Business Process Choreographer container settings > Business Process Choreographer container**.
 - c. Click **State Observers**.
 - d. Ensure that the boxes for **Audit Logging** and **Common Event Infrastructure Logging** are checked for both the **Business Flow Manager** and the **Human Task Manager**. If the check boxes are not selected, then you must select them and restart the cluster.

What to do next

If you had to select any of the boxes, then you must restart the server or cluster for the changes to take effect.

Configuring logging for service component events

You may choose to use the logging facilities of WebSphere Application Server to capture the service component events fired by process server monitoring. Use the loggers to view the data in events when you diagnose problems with the processing of your applications.

WebSphere Process Server uses the extensive logging facilities of the underlying WebSphere Application Server to allow you to capture the events fired by process server monitoring at service component event points. You can use the administrative console to specify the particular service component event points that you want to monitor, the amount of payload detail contained in the resulting service component events, and the method used to publish the results, such as to a file of a certain format, or directly to a console. Monitor logs contain events encoded in Common Base Event format, and you can use the information contained in the event elements to trace problems with the processing of your service components.

The functionality of WebSphere Application Server logging and tracing capabilities is documented in considerable detail in the WebSphere Application Server documentation, with complete details of how logging and tracing is used within the entire product. This section provides only supplemental information about logging as it relates to the service components that are specific to WebSphere Process Server. Consult the information in the WebSphere Application Server documentation for using logging and trace with other components of the entire product.

Enabling the diagnostic trace service

Use this task to enable the diagnostic trace service, which is the logging service that can manage the amount of detail contained in the service component event.

Before you begin

You must have the business process and human task containers configured to allow Common Event Infrastructure (CEI) logging and audit logging.

About this task

The diagnostic trace service is the only logger type that can provide the level of detail required to capture the detail contained in the elements of service component events. You must enable the diagnostic trace service before you start the process server in order to log events. The service must also be enabled if you use the administrative console to select service component event points for monitoring using the CEI server.

Procedure

1. In the navigation pane, click **Servers > Application Servers**.
2. Click the name of the server that you want to work with.
3. Under Troubleshooting, click **Diagnostic Trace service**.
4. Select **Enable log** on the **Configuration** tab.
5. Click **Apply**, and then **Save**.
6. Click **OK**.

What to do next

If the process server was already started, then you must restart it for the changes to take effect.

Configuring logging properties using the administrative console

Use this task to specify that the monitoring function publish service component events to a logger file.

About this task

Before WebSphere Process Server applications can log monitored events, you must specify the service component event points that you want to monitor, what level of detail you require for each event, and format of the output used to publish the events to the logs. Using the administrative console, you can:

- Enable or disable a particular event log.
- Specify the level of detail in a log.
- Specify where log files are stored, how many log files are kept, and a format for log output.

You can change the log configuration statically or dynamically. Static configuration changes affect applications when you start or restart the application server. Dynamic or run time configuration changes apply immediately.

When a log is created, the level value for that log is set from the configuration data. If no configuration data is available for a particular log name, the level for that log is obtained from the parent of the log. If no configuration data exists for the parent log, the parent of that log is checked, and so on up the tree, until a log with a non-null level value is found. When you change the level of a log, the change is propagated to the children of the log, which recursively propagates the change to their children, as necessary.

Procedure

1. Enable logging and set the output properties for a log:
2. In the navigation pane, click **Servers > Application Servers**.
3. Click the name of the server that you want to work with.

4. Under Troubleshooting, click **Logging and tracing**.
5. Click **Change Log Detail levels**.
6. The list of components, packages, and groups displays all the components that are currently registered on the running server; only process server events that have been invoked at least once will appear on this list. All process server components with event points that can be logged are listed under one of the components that start with the name **WBILocationMonitor.LOG**.
 - To select events for a static change to the configuration, click the **Configuration** tab.
 - To select events for a dynamic change to the configuration, click the **Runtime** tab.
7. Select the event or group of events that you want to log.
8. Set the logging level for each event or group of events.

Note: Only the levels FINE, FINER, and FINEST are valid for CEI event logging.

9. Click **Apply**.
10. Click **OK**.
11. To have static configuration changes take effect, stop then restart the process server.

Results

By default, the loggers will publish their output to a file called `trace.log`, located in the `install_root/profiles/profile_name/logs/server_name` folder.

Tutorial: Logging service component events

This tutorial will give you an example of how to set up service component event monitors that are published to the logger, and how to view the events in the log file. The scenario you will follow for this example will show you how to select service component event points for monitoring in applications already deployed and running on your process server. You will see how the monitoring function fires an event whenever the processing of an application reaches one of those event points. Each of those fired events takes the form of a standardized Common Base Event, which is published as an XML string directly to a log file.

Objectives of this tutorial

After completing this tutorial you will be able to:

- Select service component event points to monitor, with the output published to the process server loggers.
- View the stored events in the log files.

Time required to complete this tutorial

This tutorial requires approximately 15-20 minutes to complete.

Prerequisites

In order to perform this tutorial, you must have:

- Configured and started a process server.
- Configured Common Event Infrastructure.
- Enabled the diagnostic trace service on the process server.

- Installed and started the Samples Gallery application on the process server.
- Installed and started the business rules sample application on the process server. Follow the instructions on the Samples Gallery page to set up and run the business rules sample application.

After all of these prerequisites have been completed, run the business rules sample application from the Samples Gallery at least once before proceeding with the tutorial.

Example: Monitoring events in the logger: About this task

You will use the business rules sample application for this scenario, so you should already have the web page containing this application already open. Keep it open, since you will be running the sample after you specify monitoring parameters. Ensure that you have already run the sample at least once, so that it will appear in the list of functions that you can select to monitor.

Procedure

1. Open the administrative console.
2. In the navigation pane, click **Servers > Application Servers**.
3. Click *server_name*.
4. Under Troubleshooting, click **Logging and tracing**
5. Click **Change Log Detail levels**
6. Select the **Runtime** tab.
7. Expand the tree for **WBILocationMonitor.LOG.BR** and you will see seven event types under the **WBILocationMonitor.LOG.BR.brsmple.*** element. The first event is called **WBILocationMonitor.LOG.BR.brsmple_module.DiscountRuleGroup**, which includes a single function, named **Operation._calculateDiscount**, with the following natures:
 - ENTRY
 - EXIT
 - FAILURE
 - SelectionKeyExtracted
 - TargetFound
8. Click on each of the events and select **finest**.
9. Click **OK**.
10. Switch the business rules sample application page, and run the application once.
11. Use a text editor to open the trace.log file located in the *profile_root/logs/server_name* folder on your system.

Results

You should see lines in the log containing the business rule events fired by the monitor when you ran the sample application. The main thing you will probably notice is that the output consists of lengthy, unparsed XML strings conforming to the Common Base Event standard. Examine the ENTRY and EXIT events, and you will see that business object — which was included because you selected the **finest** level of detail — is encoded in hexadecimal format. Compare this output with events published to the Common Event Infrastructure server, which parses the XML into a readable table and decodes any business object data into a readable

format. You may want to go back through this exercise and change the level of detail from **finest** to **fine** or **finer**, and compare the differences between the events.

After completing this exercise, you should understand how to select service component event points for monitoring to the logger. You have seen that the events fired in this type monitoring have a standard format, and that the results are published as a string in raw XML format directly to a log file. Viewing the published events is simply a matter of opening the log file in a text editor, and deciphering the contents of individual events.

What to do next

If you no longer want to monitor the business rules sample application, you can go back to through the steps outlined here and reset the level of detail for the sample events to **info**.

Audit logging for business rules and selectors

You can set up WebSphere Process Server to automatically log any changes made to business rules and selectors.

You can configure your server to automatically detect when changes are made to business rules and selectors, and create an entry in a log file detailing the changes. You can choose to have the log entries written to either the standard JVM SystemOut.log file, or to a custom audit log file of your choice. Depending on how the changes are made, the process server where each business rule or selector change is made will log the:

- name of the person making the change
- location from where the change request originated
- old business rule or selector object
- new business rule or selector replacing the old object

The business rule and selector objects are the complete business rule set, decision table, business rule group, or selector for both the business rule or selector that is replaced and the new version which replaced it. You will have to examine the logs (the audit output cannot be directed to the Common Event Infrastructure database) to determine the changes that were made, by comparing the old and new business rules or selectors. The following scenarios describe the circumstance when logging occurs, if it has been configured, and the contents of the log entry:

Scenario	Result	Log entry contents
Publish business rules using the Business Rule Manager	Request	User ID, Server name (including Cell and Node, if applicable), old business rule ruleset, new ruleset.
	Failure	User ID, Server name (including Cell and Node, if applicable), old business rule ruleset, new ruleset.
Repository database update and commit (from attempt to publish using the Business Rule Manager)	Success	User ID, old ruleset, new ruleset.
	Failure	User ID, new ruleset.

Scenario	Result	Log entry contents
Exporting a selector or business rule group	Request	User ID, selector or business rule group name.
	Success	User ID, Server name (including Cell and Node, if applicable), copy of exported selector or business rule group
	Failure	User ID, Server name (including Cell and Node, if applicable), selector or business rule group name.
Importing a selector or business rule group	Request	User ID, copy of new selector or business rule group.
	Success	User ID, Server name (including Cell and Node, if applicable), copy of imported selector or business rule group, copy of selector or business rule group that was replaced by the imported version.
	Failure	User ID, Server name (including Cell and Node, if applicable), copy of selector or business rule group that was to be imported.
Application install	Success	User ID, Server name (including Cell and Node, if applicable), selector or business rule group name.
	Failure	User ID, Server name (including Cell and Node, if applicable), selector or business rule group name.
Application update (through the administrative console or wsadmin command)	Success	User ID, Server name (including Cell and Node, if applicable), copy of new selector or business rule group, copy of old selector or business rule group.
	Failure	User ID, Server name (including Cell and Node, if applicable), copy of new selector or business rule group
Previously deployed application with existing business rules, selectors or both is started	Success	Server name (including Cell and Node, if applicable), copy of selector or business rule group.
	Failure	Server name (including Cell and Node, if applicable), copy of selector or business rule group.

Monitoring service components with the Common Event Infrastructure server

You can choose to have service component monitoring results published to a Common Event Infrastructure server. Service component event points can be specified for monitoring with the Common Event Infrastructure server on a permanent basis for viewing and managing application flow, or on an ad-hoc basis for troubleshooting problems.

You can use WebSphere Process Server monitoring to publish the data in service component event points within service component events that are fired across the Common Event Infrastructure (CEI) bus. This approach to monitoring allows you much more flexibility in analyzing your service component activities on your system. You can also use browsers optimized for CEI events, such as the Common Base Event browser, which is included with the process server.

The events are structured identically to the events sent to loggers, but are stored on a database which can be accessed by viewers designed specifically for analyzing service component events. Service component event points can be specified within an application, when it is created, for continual monitoring at all times after the application is deployed and running on a server -- a method known as "static" monitoring. You would perform static monitoring on service component event points that are of particular importance in the proper flow of component processing on your system. With this information, you can easily oversee the overall actions of, and interactions between, the service component processes running on your system. You would also have the ability to quickly detect deviations from the normal flow of these processes, which may indicate that your service components are not working properly.

To configure static monitoring of service components, you would use WebSphere Integration Developer to select the service component event points in your applications that will be deployed on the process server. The selections are specified in the form of an XML file with a .mon extension that will be deployed along with the application. Once deployed on a running server, you will not be able to turn off or lower the detail level of the monitoring for events specified in the .mon file of the application; you must stop the server and undeploy the application to stop this kind of monitoring. Consult the WebSphere Integration Developer Information Center for details on creating and deploying applications with .mon files.

You can also select service component event points for "dynamic" monitoring, which can be enabled and disabled on an application already deployed to a running server. The rationale for performing dynamic monitoring using the CEI server is essentially the same as that for logging: to diagnose and troubleshoot problems on your system. The output is essentially the same as that which is published to loggers, with Common Base Event elements comprising the structure for each event fired across the CEI bus. Also, like logging data, the differences in detail levels affect only how much of the payload is encoded within the event.

Configuring service component event monitoring using the administrative console

Use the administrative console to dynamically specify the monitoring function to publish service component events to the Common Event Infrastructure server.

Before you begin

You must enable the diagnostic trace service, just as you would with the logger. After you restart your server you would invoke the events you want to monitor once, because that will cause them to appear on the list of events available for monitoring.

About this task

This method of selecting events for monitoring is used for applications that have already been deployed on a process server. Events that are specified in a .mon file that is deployed with the application on the process server are monitored by the Common Event Infrastructure (CEI) database regardless of any changes you make here. For those events, you can only specify a greater level of detail to be captured and published to the CEI database. The output that is published to the CEI database is very similar to that published by loggers.

Procedure

1. From the administrative console, click **Troubleshooting > Logging and tracing**.
2. Click **Change Log Detail levels**
3. The list of components, packages, and groups displays all the components that are currently registered on the running server; only process server events that have been invoked at least once appear on this list. All process server events that can be logged are listed under one of the components that start with the name **WBILocationMonitor.CEI**.
 - To make a static change to the configuration, click the **Configuration** tab.
 - To change the configuration dynamically, click the **Runtime** tab.
4. Select an event or group of events to monitor.
5. Click the level of detail that you want to capture for each event.

Note: Only the levels FINE, FINER, and FINEST are valid for CEI events.

6. Click **Apply**, and then **Save**.
7. Click **OK**.
8. If you made a static change to the configuration, then you will have to restart the process server for the changes to take effect.

Results

You can view the monitored event results in the Common Base Event browser.

Tutorial: Using the Common Event Infrastructure server for event monitoring

This tutorial will guide you through an example demonstrating how to set up service component event monitors that are published to the Common Event Infrastructure (CEI) server, and how to view those stored events on the Common Base Event browser. The example you will use in this scenario does not involve static monitoring, whereby an application deployed with a .mon file will continually monitor specific service components event points. For information about how to perform static monitoring, consult the IBM WebSphere Integration Developer Information Center.

The scenario you will follow for this example, instead, will show you how to select for monitoring event points on service components in applications already deployed and running on your process server. You will see how the monitoring

function fires an event whenever the processing of an application reaches one of those event points. Each of those fired events are published to the CEI server, which will store the event information about its database. You will use the Common Base Event browser to view the events.

Objectives of this tutorial

After completing this tutorial you will be able to:

- Select service component event points to monitor, with events published to the CEI server.
- View the stored events with the Common Base Event browser.

Time required to complete this tutorial

This tutorial requires approximately 15-20 minutes to complete.

Prerequisites

In order to perform this tutorial, you must have:

- Configured and started a process server.
- Configured the CEI and its database.
- Enabled the diagnostic trace service on the process server.
- Installed and started the Samples Gallery application on the process server.
- Installed and started the business rules sample application on the process server. Follow the instructions on the Samples Gallery page to set up and run the business rules sample application.

After all of these prerequisites have been completed, run the business rules sample application from the Samples Gallery at least once before proceeding with the tutorial.

Example: Monitoring with the Common Event Infrastructure server: About this task

You will use the business rules sample application for this scenario; consequently, you should already have the web page containing this application already open. Keep it open, since you will be running the sample after you specify monitoring parameters. Ensure that you have already run the sample at least once, because that will cause it to appear in the list of functions that you can select to monitor.

Procedure

1. Open the administrative console.
2. In the navigation pane, click **Servers > Application Servers**.
3. Click *server_name*.
4. Under Troubleshooting, click **Logging and tracing**
5. Click **Change Log Detail levels**
6. Select the **Runtime** tab.
7. Expand the tree for **WBILocationMonitor.CEI.BR** and you will see five event types under the **WBILocationMonitor.CEI.BR.brsample.*** element. Each event type includes the name **WBILocationMonitor.CEI.BR.brsample_module.DiscountRuleGroup**, appended by the function **Operation._calculateDiscount**, and the following natures:

- ENTRY
 - EXIT
 - FAILURE
 - SelectionKeyExtracted
 - TargetFound
8. Click on each of the events and select **finest**.
 9. Click **OK**.
 10. Switch the business rules sample application page, and run the application once.
 11. Go back to the administrative console, and select **Integration Applications > Common Base Event Browser** from the navigation pane.
 12. If you are running your server on node within a Network Deployment environment, then you may need to modify the **Event Data Store** field to include to the names of your server and node. Enter the string in the following form: 'cell/nodes/*node_name*/servers/*server_name*/ejb/com/ibm/events/access/EventAccess'.
 13. Press **Get Events**.

Results

You should now see a list in the upper pane of the Common Base Event browser of the four business rule events that were published to the CEI server when you ran the sample application. Select one of the events, and you will be shown the contents of the event in the lower pane. Compare this to the events published to the loggers. Notice that the browser has parsed the original XML string that was published to the CEI server, and that the business object code in the ENTRY and EXIT events was converted from the original hexadecimal format to readable XML. You may want to go back through this exercise and change the level of detail from **finest** to **fine** or **finer**, and compare the differences between the events.

After completing this exercise, you should understand how to select service component event points for monitoring using the CEI server. You have seen that the events fired in this type monitoring have a standard format, and that the results are published to a database. You should also be able to use the Common Base Event browser to retrieve events from the database, and view the information for individual events in a parsed table format on the browser.

What to do next

If you no longer want to monitor the business rules sample application, you can go back to through the steps outlined here and reset the level of detail for the sample events to **info**.

Session monitoring

You can monitor multiple events that are part of the same session, by using the Common Base Event browser to find all events on the Common Event Infrastructure database that contain the identical session ID attribute.

WebSphere Process Server has enhanced capabilities with which you can identify all of the service component events that are part of a single session. The standard elements for the Common Base Event include an attribute under the `contextDataElement` element, called `WBISessionID`. A unique identifier for an individual session is stored in this attribute, for all service component events that

were part of that session. You can use the **SessionID** field in the Common Base Event browser to search for events stored on the Common Event Infrastructure (CEI) database that match the session ID you specify. With this capability, you can easily review the process flow and contents of all the service component events. You can use this information to assess the efficiency of your applications, and aid you in diagnosing problems that occur only under certain circumstances.

You can use the Common Base Event browser to view the returned list of events and their associated contents. If you click the All Events view, you will see two columns named **Failed** and **Business Process**. If a particular event has a link in the **Failed** column, you can click that link to view more details about the failed event. Similarly, if there is a link in the **Business Process** associated with a particular event, you can click that link to open the Business Process Explorer and view further information about the business process or human task event.

Chapter 3. Viewing monitored events

There are a number of ways for you to view the published results of your monitored events, depending on the type of monitoring you are using. This section presents methods that you can use to view performance data, event logs, and service component events stored on a Common Event Infrastructure database

Viewing performance metrics with the Tivoli Performance Viewer

This topic explains how you can use the Tivoli Performance Viewer to start and stop performance monitoring; view Performance Monitoring Infrastructure data in chart or table form as it occurs on your system; and, optionally, log the data to a file that you can later review in the same viewer.

Before you begin

It is assumed that one or more servers have been created and are running on the node, that the Performance Monitoring Infrastructure (PMI) is enabled, and that the service component event points that you want to monitor have been invoked at least once so that they can be selected from within the viewer.

About this task

The Tivoli Performance Viewer (TPV) is a powerful application that allows you view a variety of details of all aspects of the performance of your process server. The section entitled "Monitoring performance with Tivoli Performance Viewer" in the contains details on how to use this tool for a variety of purposes, and you should consult this resource for complete instructions on using this program. This section will be limited to discussing the viewing of performance data for WebSphere Process Server-specific events.

The performance viewer enables administrators and programmers to monitor the current health of WebSphere Process Server. Because the collection and viewing of data occurs on the process server, performance is affected. To minimize performance impacts, monitor only those servers whose activity you want to monitor.

Note: When viewing these statistics, you should not mix counter-type statistics with duration-type statistics. Counters are cumulative, and the scales against which they are graphed them can quickly grow depending on your application. Duration statistics, in contrast, tend to remain within a certain range because they represent the average amount of time that it takes your system to process each event. Consequently, the disparity between the statistics and their relative scales may cause one or the other type of statistic to appear skewed in the viewer graph.

• View current performance activity

1. Click **Monitoring and Tuning > Performance Viewer > Current Activity** in the administrative console navigation tree.
2. Select **Server**, and click the name of the server whose activity you want to monitor. You can alternatively select the check box for the server whose activity you want to monitor, and click **Start Monitoring**. To start monitoring multiple servers at the same time, select the servers and click **Start Monitoring**.

3. Select **Performance Modules**.
4. Select the check box beside the name of each performance module that you want to view. All process server-specific events that emit performance statistics, and that have been invoked at least once, are listed under the **WBStats.RootGroup** hierarchy. Expand the tree by clicking + next to a node and shrink it by clicking – next to a node.
5. Click on **View Modules**. A chart or table providing the requested data is displayed on the right side of the page. Charts are displayed by default. Each module has several counters associated with it. These counters are displayed in a table underneath the data chart or table. Selected counters are displayed in the chart or table. You can add or remove counters from the chart or table by selecting or clearing the check box next to them. By default, the first three counters for each module are shown.
You can select up to 20 counters and display them in the TPV in the **Current® Activity** mode.
6. Optional: To remove a module from a chart or table, clear the check box next to the module and click **View Modules** again.
7. Optional: To view the data in a table, click **View Table** on the counter selection table. To toggle back to a chart, click **View Graph**.
8. Optional: To view the legend for a chart, click **Show Legend**. To hide the legend, click **Hide Legend**.
9. When you have finished monitoring the performance of your events, click on **Tivoli Performance Viewer**, select the server you were monitoring, and click **Stop Monitoring**.

- **Log performance statistics**

While monitoring is active on a server, you can log the data from all the PMI counters that are currently enabled and record the results in a TPV log file. You can view the TPV log file for a particular time period multiple times, selecting different combinations of up to 20 counters each time. You have the flexibility to observe the relationships among different performance measures in the server during a particular period of time.

1. Click on **Start Logging** when viewing summary reports or performance modules.
2. When finished, click **Stop Logging**. By default, the log files are stored in the *profile_root/logs/tpv* directory on the node on which the server is running. The TPV automatically compresses the log file when it finishes writing to it to conserve space. At this point, there must only be a single log file in each compressed file and it must have the same name as the compressed file.
3. Click **Monitoring and Tuning > Performance Viewer > View Logs** in the administrative console navigation tree to view the logs

Viewing and interpreting service component event log files

This topic discusses how you would interpret the information in a log file generated by service component monitoring. You can view the log files in the log viewer on the administrative console, or in a separate text file editor of your choice.

Events fired to the logger by service component monitoring are encoded in Common Base Event format. When published to a log file, the event is included as a single, lengthy line of text in XML tagging format, which will also include several logger-specific fields, as outlined below. You should consult the event catalog section of this documentation for details on deciphering the Common Base

Event coding of the logged event. Use this section to understand the other fields contained in each entry of the log file, and how the format you chose for the log file when you configured the logger is structured.

Basic and advanced format fields

On a process server, logging output can be directed either to a file or to an in-memory circular buffer. If trace output is directed to the in-memory circular buffer, it must be dumped to a file before it can be viewed. Output is generated as plain text in either basic, advanced or log analyzer format as specified by the user. The basic and advanced formats for output are similar to the basic and advanced formats that are available for the message logs. Basic and Advanced Formats use many of the same fields and formatting techniques. The fields that can be used in these formats include:

TimeStamp

The timestamp is formatted using the locale of the process where it is formatted. It includes a fully qualified date (YYMMDD), 24 hour time with millisecond precision and the time zone.

ThreadId

An 8 character hexadecimal value generated from the hash code of the thread that issued the trace event.

ThreadName

The name of the Java thread that issued the message or trace event.

ShortName

The abbreviated name of the logging component that issued the trace event. This is typically the class name for WebSphere Process Server internal components, but may be some other identifier for user applications.

LongName

The full name of the logging component that issued the trace event. This is typically the fully qualified class name for WebSphere Process Server internal components, but may be some other identifier for user applications.

EventType

A one character field that indicates the type of the trace event. Trace types are in lower case. Possible values include:

- 1 a trace entry of type fine or event.
- 2 a trace entry of type finer.
- 3 a trace entry of type finest, debug or dump.
- Z a placeholder to indicate that the trace type was not recognized.

ClassName

The class that issued the message or trace event.

MethodName

The method that issued the message or trace event.

Organization

The organization that owns the application that issued the message or trace event.

Product

The product that issued the message or trace event.

Component

The component within the product that issued the message or trace event.

Basic format

Trace events displayed in basic format use the following format:

```
<timestamp><threadId><shortName><eventType>[className] [methodName] <textmessage>
    [parameter 1]
    [parameter 2]
```

Advanced format

Trace events displayed in advanced format use the following format:

```
<timestamp><threadId><eventType><UOW><source=longName>[className] [methodName]
<Organization><Product><Component>[thread=threadName]
<textMessage>[parameter 1=parameterValue] [parameter 2=parameterValue]
```

Log analyzer format

Specifying the log analyzer format allows you to open trace output using the Log Analyzer tool, which is an application included with WebSphere Application Server. This is useful if you are trying to correlate traces from two different server processes, because it allows you to use the merge capability of the Log Analyzer.

Viewing events with the Common Base Event browser

Use the Common Base Event browser to select, sort, and view events.

Before you begin

This task assumes you are logged into the server administrative console.

About this task

The event browser uses the event access interface to query event data. The results of the query are shown in the browser.

Procedure

1. Begin by opening the event browser. Click **Integration Applications** and then **Common Base Event Browser** in the navigation pane of the administrative console.
2. Specify the events you want to view.
3. Select the view of the returned events.
4. In any of the browser panels, when you have finished selecting search or sort criteria, click the **Get Events** button at the bottom of the browser panel to display the events.

Specifying the events to view

How to use the Common Base Event browser to specify search criteria for querying events in the event database.

Before you begin

This task assumes that you have already opened the event browser and are viewing the Get Events panel.

The Event Data Store Properties fields require completion. The Event Filter Properties fields are optional, and you can narrow your events search based on time, date, server name, sub-component name, and event severity parameters.

Procedure

1. Required: Specify the Event Data Store to search.
The field is a Java Naming and Directory Interface (JNDI) name, an Enterprise JavaBeans (EJB) reference that can be configured in the administrative console. The server default is `java:comp/env/eventsaccess`, but you may need to specify a different name if you are working in a Network Deployment environment (see the WebSphere Application Server Network Deployment, version 6.1 documentation for more details about JNDI naming).
2. Required: Specify the Event Group to search.
This is the event group from which events are retrieved. The default group is All events.
3. Required: Specify the number of events to retrieve.
The maximum number of events to search is 500.
4. Optional: Specify the Creation Date (calendar period) for the report.
Enter the start and end dates.
5. Optional: Specify the Creation Time (time period) for the report.
Enter the start and end times.
6. Optional: Specify the server name.
7. Optional: Specify the sub-component name, if applicable.
8. Optional: Specify an event's priority. The range of events priorities to retrieve is from 0 (lowest priority) to 100 (highest priority).
9. Optional: Specify an event's severity.
The range of events severities to retrieve is from 0 (least severe) to 70 (most severe).
10. Click **Get Events**.

Results

The number of Common Base Events matching the search criteria is displayed. If the results you queried are not displayed, see the "Troubleshooting the Common Base Event browser" topic in the *Troubleshooting and Support* PDF file.

To view the returned events, select a view from the navigation bar. You can click **All Events**, **BPEL Process Events**, **User Data Events**, or **Server Events**. When you view event data, you can change your search criteria at any time by clicking **Get Events**.

What to do next

After events are returned, you can work with them to get various levels of event detail.

Working with events returned from the event browser

You use the event browser to view the events returned from a query.

Before you begin

This task acts on data that is returned by a submitted query, as described in the topic *Specifying the events to view*.

About this task

The query returns all the events that meet your criteria.

Procedure

1. Click a view from the navigation bar.

The navigation bar offers the following views of the returned query:

All Events

All the events returned.

BPEL Process Events

Business Process Choreographer events for a specific process instance.

User Data Events

Events with the extension name ECS:UserDataEvent. This event type is created by the addUserDataEvent method of the ECSEmitter class.

Server Events

Events for a specific server.

2. Perform one of the following actions.
 - If you click **BPEL Process Events** in step 1, you must click a process template, and then a process instance.
 - If you click **Server Events** in step 1, you must click a server.
3. Click an event, to display the event data in the pane at the bottom of the browser window.

Troubleshooting the Common Base Event browser

There are four primary conditions under which you are unable to access the Common Base Event browser.

Conditions

“Cannot find server”

The server is unavailable. When you attempt to launch the event browser URI, a “Cannot find server” browser page will be returned, which indicates that the server is unavailable. In this case, you need to contact the server administrator to determine the cause of the problem.

“File not found”

The server is available; however, the event browser application may not be installed or started on the server. When you attempt to launch the event browser URI, a “File not found” browser page will be returned, which indicates that the server is available, but the URI is not available on that server. In this case, you need to contact the server administrator to determine the cause of the problem.

Logon panel appears

The server and the event browser are available; however, you have not been mapped to the proper role to allow access to the event browser. You will be prompted with a logon panel. When you enter your userID and password, attempting to log in, the login will fail. In this case, you need to contact the server administrator to get the proper authorization to launch the event browser.

Error message on “Get event data” panel

The server and the event browser are available, and you have the proper authority to gain access; however, the Common Event Infrastructure server is unavailable. An error message will be displayed on the event browser **Get Events** panel, when you click the **Get Events** button. The error information is logged to the message log.

Chapter 4. Event catalog

The event catalog contains the specifications for all the events that can be monitored for each service component type, as well as the associated Common Base Event extended data elements produced by each event. You should use the information presented in this section as reference material that will enable you to understand how individual events are structured. This knowledge will help you decipher the information contained in each event, so that you can quickly identify the pieces of information you need from the relatively large amount of data generated by each event. The information included in this section covers the structure and standard elements of the Common Base Event; the list of events for the Business Process Choreographer service components and WebSphere Process Server-specific service components; and, the extensions to the Common Base Event unique to each event type. There is also a discussion of how business objects that may be processed by a service component are captured in service component events.

When an event of a given type is fired across the Common Event Infrastructure (CEI) bus to the CEI server or to a logger, it takes the form of a Common Base Event — which is, essentially, an XML encapsulation of the event elements created according to the event catalog specification. The Common Base Event includes a set of standard elements, process server component identification elements, Event Correlation Sphere identifiers, and additional elements unique to each event type. All of these elements are passed to the CEI server or logger whenever an event is fired by a service component monitor, with one exception: if the event includes the business object code within the payload, you may specify the amount of business object data that you want to include in event.

The Common Base Event standard elements

The elements of the Common Base Event that are included in all events fired from service component monitoring are listed here.

Attribute	Description
version	Set to 1.0.1.
creationTime	The time at which the event is created, in UTC.
globalInstanceId	The identifier of the Common Base Event instance. This ID is automatically generated.
localInstanceId	This ID is automatically generated (may be blank).
severity	The impact that the event has on business processes or on human tasks. This attribute is set to 10 (information). Otherwise, it is not used.
priority	Not used.
reporterComponentId	Not used.
locationType	Set to Hostname.
location	Set to the host name of the executing server.
application	Not used.
executionEnvironment	A string that identifies the operating system.

Attribute	Description
component	Process server version. For business processes and human tasks: Set to WPS#, followed by the identification of the current platform and the version identification of the underlying software stack.
componentType	The component QName, based on the Apache QName format. For business processes, set to: www.ibm.com/namespaces/autonomic/Workflow_Engine For human tasks, set to: www.ibm.com/xmlns/prod/websphere/scdl/human-task
subComponent	The observable element name. For business processes, set to BFM. For human tasks, set to HTM.
componentIdType	set to ProductName.
instanceId	The identifier of the server. This identifier has the format <i>cell_name/node_name/server_name</i> . The delimiters are operating system dependent.
processId	The process identifier of the operating system.
threadId	The thread identifier of the Java virtual machine (JVM).
Situation Type	The type of situation that caused the event to be reported. For process server-specific components, set to ReportSituation.
Situation Category	The category of the type of situation that caused the event to be reported. For process server-specific components, set to STATUS.
Situation Reasoning Scope	The scope of the impact of the situation reported. For process server-specific components, set to EXTERNAL.
ECSCurrentID	The value of the current Event Correlation Sphere ID.
ECSParentID	The value of the parent Event Correlation Sphere ID.
WBISessionID	The value of the current Session ID.
extensionName	Set to the event name.

Business objects in events

Business object data is, starting with version 6.1, carried within the event in XML format. The Common Base Event format includes an `xs:any` schema, which encapsulates the business object payload in XML elements.

You specify the level of business object detail that will be captured in service component events. This level of detail affects only the amount of business object code that will be passed to the event; all of the other Common Base Event elements (both standard and event-specific) will be published to the event. The names of the detail levels applicable to service component events differ depending on whether you created a static monitor using WebSphere Integration Developer, or a dynamic monitor on the administrative console, but they correspond as shown in the table below:

Administrative console detail level	Common Base Event/WebSphere Integration Developer detail level	Payload information published
FINE	EMPTY	None.
FINER	DIGEST	Payload description only.
FINEST	FULL	All of the payload.

The detail level is specified by PayloadType element which is part of the event instance data. The actual business object data is included in the event only if the monitor is set to record FULL/FINEST detail. The business object data itself is included in the Common Base Event under an `xsd:any` schema. You will actually see the process server business object payloads with the root element named `wbi:event`. If you are publishing the event output to the logger, then you will see the output when you view the log files. If the event is published to the CEI server, then you can use the Common Base Event browser to view the event. You can then click the `wbi:event` link to view the business object data.

Business Process Choreographer events

WebSphere Process Server incorporates the Business Process Choreographer service components for business processes and human tasks. The event points that can be monitored in these components are described in this section.

Monitoring business process events

Events that are emitted on behalf of business processes consist of situation-independent data and data that is specific to business process events. The attributes and elements that are specific to business process events are described.

Business process events can have the following categories of event content.

Event data specific to business processes

In business processes, events relate to processes, activities, scopes, links, and variables.

For Business Process Choreographer version 6.1 two event formats can occur:

WebSphere Business Monitor 6.0.2 format

WebSphere Business Monitor 6.0.2 format events occur when there are processes modeled within WebSphere Integration Developer 6.0.2, or if the WebSphere Business Monitor 6.0.2 format mode is enabled within WebSphere Integration Developer 6.1. If not specified otherwise, the object-specific content for these events is written as *extendedDataElement* XML elements of the type string.

WebSphere Business Monitor 6.1 format

WebSphere Business Monitor 6.1 format events occur when there are processes modeled within WebSphere Integration Developer 6.1, and the WebSphere Business Monitor 6.1 format mode is enabled. The object-specific content for these events is written as XML elements in the `xs:any` slot in the `eventPointData` folder of the Common Base Event, and the payload message is written to the *applicationData* section. The structure of the XML is defined in the XML Schema Definition (XSD) file `BFMEvents.xsd`. The file can be found in the `install_root\ProcessChoreographer\client` directory.

Extension names for business process events

The extension name indicates the payload of the event. A list of all the extension names for business process events and their corresponding payload can be found here.

The extension name contains the string value that is used as the value of the *extensionName* attribute of the Common Base Event. This is also the name of the XML element that provides additional data about the event. The names of event elements are in uppercase, for example, BPC.BFM.BASE, and the names of XML elements are in mixed case, for example, *BPCEventCode*. Except where indicated, all data elements are of the type string.

The following extension names are available for business process events:

- “BPC.BFM.BASE”
- “BPC.BFM.PROCESS.BASE” on page 51
- “BPC.BFM.PROCESS.STATUS” on page 52
- “BPC.BFM.PROCESS.START” on page 52
- “BPC.BFM.PROCESS.FAILURE” on page 52
- “BPC.BFM.PROCESS.CORREL” on page 52
- “BPC.BFM.PROCESS.WISTATUS” on page 52
- “BPC.BFM.PROCESS.WITRANSFER” on page 53
- “BPC.BFM.PROCESS.ESCALATED” on page 53
- “BPC.BFM.PROCESS.EVENT” on page 53
- “BPC.BFM.PROCESS.PARTNER” on page 54
- “BPC.BFM.PROCESS.CUSTOMPROPERTYSET” on page 54
- “BPC.BFM.ACTIVITY.BASE” on page 54
- “BPC.BFM.ACTIVITY.STATUS” on page 56
- “BPC.BFM.ACTIVITY.FAILURE” on page 56
- “BPC.BFM.ACTIVITY.MESSAGE” on page 56
- “BPC.BFM.ACTIVITY.CLAIM” on page 56
- “BPC.BFM.ACTIVITY.WISTATUS” on page 57
- “BPC.BFM.ACTIVITY.WITRANSFER” on page 57
- “BPC.BFM.ACTIVITY.FOREACH” on page 57
- “BPC.BFM.ACTIVITY.ESCALATED” on page 57
- “BPC.BFM.ACTIVITY.EVENT” on page 57
- “BPC.BFM.ACTIVITY.CUSTOMPROPERTYSET” on page 58
- “BPC.BFM.LINK.STATUS” on page 58
- “BPC.BFM.VARIABLE.STATUS” on page 58

BPC.BFM.BASE

BPC.BFM.BASE inherits the XML elements from WBIMonitoringEvent.

Table 21. XML elements for BPC.BFM.BASE

XML element	Description
<i>BPCEventCode</i>	The Business Process Choreographer event code that identifies the event nature.
<i>processTemplateName</i>	The name of the process template. This name can differ from the display name.

Table 21. XML elements for BPC.BFM.BASE (continued)

XML element	Description
<i>processTemplateValidFrom</i>	The valid from attribute of the process template.
<i>eventProgressCounter</i>	<p>The event progress counter is used to indicate the position of the current navigation step in the execution order of all navigation steps of the same process instance.</p> <p>The event progress counter is required for long-running processes, and it can be used together with the event local counter to recreate the (possibly incomplete) order of the events belonging to the same process instance. In microflows, the event progress counter is set to zero.</p>
<i>eventLocalCounter</i>	The local counter is used to discover the order of two events that occur in the same transaction. For a microflow instance, this counter reconstructs an order of all the emitted events. For long-running processes, the local counter indicates an order in the current navigation transaction.

BPC.BFM.PROCESS.BASE

BPC.BFM.PROCESS.BASE inherits the XML elements from “BPC.BFM.BASE” on page 50.

Table 22. XML elements for BPC.BFM.PROCESS.BASE

XML element	Description
<i>processInstanceExecutionState</i>	<p>The current execution state of the process in the following format: <state code>-<state name>. This attribute can have one of the following values:</p> <ul style="list-style-type: none"> 1 - STATE_READY 2 - STATE_RUNNING 3 - STATE_FINISHED 4 - STATE_COMPENSATING 5 - STATE_FAILED 6 - STATE_TERMINATED 7 - STATE_COMPENSATED 8 - STATE_TERMINATING 9 - STATE_FAILING 10 - STATE_INDOUBT 11 - STATE_SUSPENDED 12 - STATE_COMPENSATION_FAILED
<i>processTemplateId</i>	The ID of the process template.
<i>processInstanceDescription</i>	The description of the process instance.
<i>principal</i>	The name of the user who is associated with this event.

BPC.BFM.PROCESS.STATUS

BPC.BFM.PROCESS.STATUS inherits the XML elements from "BPC.BFM.PROCESS.BASE" on page 51.

BPC.BFM.PROCESS.START

BPC.BFM.PROCESS.START inherits the XML elements from "BPC.BFM.PROCESS.BASE" on page 51.

Table 23. XML elements for BPC.BFM.PROCESS.START

XML element	Description
<i>username</i>	The name of the user who requested the start or restart of the process.

BPC.BFM.PROCESS.FAILURE

BPC.BFM.PROCESS.FAILURE inherits the XML elements from "BPC.BFM.PROCESS.BASE" on page 51.

Table 24. XML elements for BPC.BFM.PROCESS.FAILURE

XML element	Description
<i>processFailedException</i>	The exception message that lead to the failure of the process.

BPC.BFM.PROCESS.CORREL

BPC.BFM.PROCESS.CORREL inherits the XML elements from "BPC.BFM.PROCESS.BASE" on page 51.

Table 25. XML elements for BPC.BFM.PROCESS.CORREL

XML element	Description
<i>correlationSet</i>	The correlation set instance, in the following format: <pre><?xml version="1.0"?> <correlationSet name="correlation set name"> <property name="property name" value="property value"/>* </correlationSet></pre>

BPC.BFM.PROCESS.WISTATUS

BPC.BFM.PROCESS.WISTATUS inherits the XML elements from "BPC.BFM.PROCESS.BASE" on page 51.

Table 26. XML elements for BPC.BFM.PROCESS.WISTATUS

XML element	Description
<i>username</i>	The names of the users with work items that were created or deleted.

BPC.BFM.PROCESS.WITRANSFER

BPC.BFM.PROCESS.WITRANSFER inherits the XML elements from “BPC.BFM.PROCESS.BASE” on page 51.

Table 27. XML elements for BPC.BFM.PROCESS.WITRANSFER

XML element	Description
<i>current</i>	The user name of the current owner of the work item. This is the user whose work item has been transferred to someone else.
<i>target</i>	The user name of the new owner of the work item.

BPC.BFM.PROCESS.ESCALATED

BPC.BFM.PROCESS.ESCALATED inherits the XML elements from “BPC.BFM.PROCESS.BASE” on page 51.

Table 28. XML elements for BPC.BFM.PROCESS.ESCALATED

XML element	Description
<i>escalationName</i>	The name of the escalation.
<i>operation</i>	This is the operation that is associated with the event handler for which the inline invocation task is escalated.
<i>portTypeName</i>	The port type name of the operation that is associated with the event handler for which the inline invocation task is escalated.
<i>portTypeNamespace</i>	The port type namespace of the operation that is associated with the event handler for which the inline invocation task is escalated.

BPC.BFM.PROCESS.EVENT

BPC.BFM.PROCESS.EVENT inherits the XML elements from “BPC.BFM.PROCESS.BASE” on page 51.

Table 29. XML elements for BPC.BFM.PROCESS.EVENT

XML element	Description
<i>message</i> or <i>message_BO-</i>	<p>The input message or the output message for the service as a String or business object (BO) representation. The format depends on whether the Monitor Compatible Events option was selected on the Event Monitor tab in WebSphere Integration Developer.</p> <p>This attribute is only used for WebSphere Business Monitor 6.0.2 format events. For WebSphere Business Monitor 6.1 format events, the content of the message is written to the <i>applicationData</i> section, which contains one content element with the name set to the name of the message.</p>

Table 29. XML elements for BPC.BFM.PROCESS.EVENT (continued)

XML element	Description
<i>operation</i>	Name of the operation for the received event.
<i>portTypeName</i>	The port type name of the operation that is associated with the event handler.
<i>portTypeNamespace</i>	The port type namespace of the operation that is associated with the event handler.

BPC.BFM.PROCESS.PARTNER

BPC.BFM.PROCESS.PARTNER inherits the XML elements from “BPC.BFM.PROCESS.BASE” on page 51.

Table 30. XML elements for BPC.BFM.PROCESS.PARTNER

XML element	Description
<i>partnerLinkName</i>	The name of the partner link.

BPC.BFM.PROCESS.CUSTOMPROPERTYSET

BPC.BFM.PROCESS.CUSTOMPROPERTYSET inherits the XML elements from “BPC.BFM.PROCESS.BASE” on page 51.

Table 31. XML elements for BPC.BFM.PROCESS.CUSTOMPROPERTYSET

XML element	Description
<i>propertyName</i>	The name of the custom property.
<i>propertyValue</i>	The value of the custom property.
<i>associatedObjectID</i>	The ID of the associated object that is the process instance ID.
<i>associatedObjectName</i>	The name of the associated object that is the process template name.

BPC.BFM.ACTIVITY.BASE

BPC.BFM.ACTIVITY.BASE inherits the XML elements from “BPC.BFM.BASE” on page 50.

Table 32. XML elements for BPC.BFM.ACTIVITY.BASE

XML element	Description
<i>activityKind</i>	<p>The activity kind, for example, sequence or invoke. The format is: <code><kind code>-<kind name></code>. This attribute can have one of the following values:</p> <ul style="list-style-type: none"> 3 - KIND_EMPTY 21 - KIND_INVOKE 23 - KIND_RECEIVE 24 - KIND_REPLY 25 - KIND_THROW 26 - KIND_TERMINATE 27 - KIND_WAIT 29 - KIND_COMPENSATE 30 - KIND_SEQUENCE 32 - KIND_SWITCH 34 - KIND_WHILE 36 - KIND_PICK 38 - KIND_FLOW 40 - KIND_SCOPE 42 - KIND_SCRIPT 43 - KIND_STAFF 44 - KIND_ASSIGN 45 - KIND_CUSTOM 46 - KIND_RETHROW 47 - KIND_FOR_EACH_SERIAL 48 - KIND_FOR_EACH_PARALLEL 1000 - SQLSnippet 1001 - RetrieveSet 1002 - InvokeInformationService 1003 - AtomicSQLSnippetSequence
<i>state</i>	<p>The current state of the activity instance in the format: <code><state code>-<state name></code>. This attribute can have one of the following values:</p> <ul style="list-style-type: none"> 1 - STATE_INACTIVE 2 - STATE_READY 3 - STATE_RUNNING 4 - STATE_SKIPPED 5 - STATE_FINISHED 6 - STATE_FAILED 7 - STATE_TERMINATED 8 - STATE_CLAIMED 9 - STATE_TERMINATING 10 - STATE_FAILING 11 - STATE_WAITING 12 - STATE_EXPIRED 13 - STATE_STOPPED
<i>bpelId</i>	<p>The wpc:id attribute of the activity in the BPEL file. It is unique for activities in a process model.</p>
<i>activityTemplateName</i>	<p>The name of the activity template. this can differ from the display name.</p>
<i>activityTemplateId</i>	<p>The internal ID of the activity template.</p>
<i>activityInstanceDescription</i>	<p>The description of the activity instance.</p>

Table 32. XML elements for BPC.BFM.ACTIVITY.BASE (continued)

XML element	Description
<i>principal</i>	The name of the user who claimed the activity.

BPC.BFM.ACTIVITY.STATUS

BPC.BFM.ACTIVITY.STATUS inherits the XML elements from “BPC.BFM.ACTIVITY.BASE” on page 54.

BPC.BFM.ACTIVITY.FAILURE

BPC.BFM.ACTIVITY.FAILURE inherits the XML elements from “BPC.BFM.ACTIVITY.BASE” on page 54.

Table 33. XML elements for BPC.BFM.ACTIVITY.FAILURE

XML element	Description
<i>activityFailedException</i>	The exception that caused the activity to fail.

BPC.BFM.ACTIVITY.MESSAGE

BPC.BFM.ACTIVITY.MESSAGE inherits the XML elements from “BPC.BFM.ACTIVITY.BASE” on page 54.

Table 34. XML elements for BPC.BFM.ACTIVITY.MESSAGE

XML element	Description
<i>message</i> or <i>message_BO</i>	<p>The input or the output message for the service as a string or business object (BO) representation. The format depends on whether the Monitor Compatible Events option was selected on the Event Monitor tab in WebSphere Integration Developer.</p> <p>This attribute is only used for WebSphere Business Monitor 6.0.2 format events. For WebSphere Business Monitor 6.1 format events, the content of the message is written to the <i>applicationData</i> section, which contains one content element with the name set to the name of the message.</p>

BPC.BFM.ACTIVITY.CLAIM

BPC.BFM.ACTIVITY.CLAIM inherits the XML elements from “BPC.BFM.ACTIVITY.BASE” on page 54.

Table 35. XML elements for BPC.BFM.ACTIVITY.CLAIM

XML element	Description
<i>username</i>	The name of the user for whom the task has been claimed.

BPC.BFM.ACTIVITY.WISTATUS

BPC.BFM.ACTIVITY.WISTATUS inherits the XML elements from “BPC.BFM.ACTIVITY.BASE” on page 54.

Table 36. XML elements for BPC.BFM.ACTIVITY.WISTATUS

XML element	Description
<i>username</i>	The names of the users who are associated with the work item.

BPC.BFM.ACTIVITY.WITRANSFER

BPC.BFM.ACTIVITY.WITRANSFER inherits the XML elements from “BPC.BFM.ACTIVITY.BASE” on page 54.

Table 37. XML elements for BPC.BFM.ACTIVITY.WITRANSFER

XML element	Description
<i>current</i>	The user name of the current owner of the work item. This is the user whose work item has been transferred to someone else.
<i>target</i>	The user name of the new owner of the work item.

BPC.BFM.ACTIVITY.FOREACH

BPC.BFM.ACTIVITY.FOREACH inherits the XML elements from “BPC.BFM.ACTIVITY.BASE” on page 54.

Table 38. XML elements for BPC.BFM.ACTIVITY.FOREACH

XML element	Description
<i>parallelBranchesStarted</i>	The number of branches started.

BPC.BFM.ACTIVITY.ESCALATED

BPC.BFM.ACTIVITY.ESCALATED inherits the XML elements from “BPC.BFM.ACTIVITY.BASE” on page 54.

Table 39. XML elements for BPC.BFM.ACTIVITY.ESCALATED

XML element	Description
<i>escalationName</i>	The name of the escalation.
<i>operation</i>	This is the operation that is associated with the event handler for which the inline invocation task is escalated.

BPC.BFM.ACTIVITY.EVENT

BPC.BFM.ACTIVITY.EVENT inherits the XML elements from “BPC.BFM.ACTIVITY.BASE” on page 54.

Table 40. XML elements for BPC.BFM.ACTIVITY.EVENT

XML element	Description
<i>operation</i>	The name of the operation for the received event.

BPC.BFM.ACTIVITY.CUSTOMPROPERTYSET

BPC.BFM.ACTIVITY.CUSTOMPROPERTYSET inherits the XML elements from “BPC.BFM.ACTIVITY.BASE” on page 54.

Table 41. XML elements for BPC.BFM.ACTIVITY.CUSTOMPROPERTYSET

XML element	Description
<i>propertyName</i>	The name of the custom property.
<i>propertyValue</i>	The value of the custom property.
<i>associatedObjecID</i>	The ID of the associated object that is the activity instance ID.
<i>associatedObjectName</i>	The name of the associated object that is the activity template name.

BPC.BFM.LINK.STATUS

BPC.BFM.LINK.STATUS inherits the XML elements from “BPC.BFM.BASE” on page 50.

Table 42. XML elements for BPC.BFM.LINK.STATUS

XML element	Description
<i>elementName</i>	The name of the link.
<i>description</i>	The description of the link.
<i>flowBpelId</i>	The ID of the flow activity where the link is defined.

BPC.BFM.VARIABLE.STATUS

BPC.BFM.VARIABLE.STATUS inherits the XML elements from “BPC.BFM.BASE” on page 50.

Table 43. XML elements for BPC.BFM.VARIABLE.STATUS

XML element	Description
<i>variableName</i>	The name of the variable.

Table 43. XML elements for BPC.BFM.VARIABLE.STATUS (continued)

XML element	Description
<i>variableData</i> or <i>variableData_BO</i>	<p>If the variable <i>variableName</i> is not initialized, there is no <i>variableData</i> or <i>VariableData_BO</i> element. The variable's data is represented either as a String or business object (BO). The format depends on whether the Monitor Compatible Events option was selected on the Event Monitor tab in WebSphere Integration Developer.</p> <p>This attribute is only used for WebSphere Business Monitor 6.0.2 format events. For WebSphere Business Monitor 6.1 format events, the content of the variable is written to the <i>applicationData</i> section, which contains one content element with the name set to the name of the variable.</p>
<i>bpellid</i>	The Business Process Choreographer ID for the variable.
<i>principal</i>	The name of the user who updated the variable.

Business process events

Business process events are sent if monitoring is requested for the business process elements in WebSphere Integration Developer. A list of all the events that can be emitted by business processes can be found here.

An event is emitted when the state of a process or activity changes. The following types of events can be caused by business process:

- "Process events" on page 60
- "Activity events" on page 62
- "Activity scope events" on page 65
- "Link events" on page 65
- "Variable events" on page 65

XML Schema Definition (XSD) files

The event structure is described in the XML Schema Definition (XSD) file *BFMEvents.xsd*. The file can be found in the *install_root*\ProcessChoreographer\client directory.

Key to table columns

The columns in the following tables contain:

Code Contains the number of the event. For WebSphere Business Monitor 6.0.2 format events, the value is written to the Common Base Event as an extended data element with the name *BPCEventCode*. For WebSphere Business Monitor 6.1 format events, the value is written to the *xs:any* slot of the Common Base Event.

Extension name

The *extensionName* contains a string value which defines the event specific information that is contained in the Common Base Event. This is also the

name of the XML element that provides additional data about the event. For further information on extension names, see “Extension names for business process events” on page 50.

Situation

Refers to the situation name of the business process event. For details of situations, see “Situations in business process events” on page 66.

Event nature

A pointer to the event situation for a business process element in the EventNature parameter, as they are displayed in WebSphere Integration Developer.

Process events

The following table describes all process events.

Code	Extension name	Situation	Event nature	Description
21000	BPC.BFM.PROCESS.START	Start	ENTRY	Process started
21001	BPC.BFM.PROCESS.STATUS	Report	SUSPENDED	Process suspended. To suspend process instances use the Business Process Choreographer Explorer.
21002	BPC.BFM.PROCESS.STATUS	Report	RESUMED	Process resumed. Only suspended processes can be resumed. To resume process instances use the Business Process Choreographer Explorer.
21004	BPC.BFM.PROCESS.STATUS	Stop	EXIT	Process completed
21005	BPC.BFM.PROCESS.STATUS	Stop	TERMINATED	Process terminated. To terminate process instances use the Business Process Choreographer Explorer.
21019	BPC.BFM.PROCESS.START	Report	RESTARTED	Process restarted
21020	BPC.BFM.PROCESS.STATUS	Destroy	DELETED	Process deleted
42001	BPC.BFM.PROCESS.FAILURE	Fail	FAILED	Process failed
42003	BPC.BFM.PROCESS.STATUS	Report	COMPENSATING	Process compensating. To compensate process instances use Business Process Choreographer Explorer.
42004	BPC.BFM.PROCESS.STATUS	Stop	COMPENSATED	Process compensated

Code	Extension name	Situation	Event nature	Description
42009	BPC.BFM.PROCESS.STATUS	Report	TERMINATING	Process terminating
42010	BPC.BFM.PROCESS.STATUS	Report	FAILING	Process failing
42027	BPC.BFM.PROCESS.CORREL	Report	CORRELATION	Correlation set initialized. Emitted when a new correlation set for the process instance is initialized. This is for example the case, when a receive activity with an initiating correlation set receives a message.
42041	BPC.BFM.PROCESS. WISTATUS	Report	WI_DELETED	Process work item deleted
42042	BPC.BFM.PROCESS. WISTATUS	Report	WI_CREATED	Process work item created
42046	BPC.BFM.PROCESS.STATUS	Fail	COMPFAILED	Process compensation failed
42047	BPC.BFM.PROCESS.EVENT	Report	EV_RECEIVED	Process event received. To define the event use the process interface. The event is generated when an event handler that is associated with a process is activated.
42049	BPC.BFM.PROCESS.ESCALATED	Report	EV_ESCALATED	Process event escalated. This event is generated when an inline invocation task is escalated, that is defined on the process level and associated with an onEvent event handler.
42056	BPC.BFM.PROCESS. WITRANSFER	Report	WI_TRANSFERRED	Process work item transferred
42058	BPC.BFM.PROCESS.PARTNER	Report	PA_CHANGE	Process partner changed. This event is generated when a new endpoint reference is assigned to a partner link.

Code	Extension name	Situation	Event nature	Description
42059	BPC.BFM.PROCESS. CUSTOMPROPERTYSET	Report	CP_SET	Process custom property set. This event is generated when a custom property of a process instance is changed.

For process events, the following event correlation sphere identifiers have the following content:

- The ECSCurrentID provides the ID of the process instance.
- The ECSParentID provides the value of the ECSCurrentID before the process instance start event of the current process.

Activity events

The following table describes all activity events.

Code	Extension name	Situation	Event nature	Description
21006	BPC.BFM.ACTIVITY.MESSAGE	Start	CREATED	Activity ready. This event is generated when a human task activity is started.
21007	For invoke activities: BPC.BFM.ACTIVITY.MESSAGE. For all other activity types: BPC.BFM.ACTIVITY.STATUS	Start	ENTRY	Activity started. For Invoke activities, business object payload is available.
21011	For invoke, human task, receive, and reply activities: BPC.BFM.ACTIVITY.MESSAGE. For pick activities: BPC.BFM.ACTIVITY.EVENT. For all other activity types: BPC.BFM.ACTIVITY.STATUS	Stop	EXIT	Activity completed. For invoke, human task, receive, and reply activities, business object payload is available.
21021	BPC.BFM.ACTIVITY.STATUS	Report	DEASSIGNED	Claim canceled. This event is generated when the claim for a human task activity is canceled..
21022	BPC.BFM.ACTIVITY.CLAIM	Report	ASSIGNED	Activity claimed. This event is generated when a human task activity is claimed.

Code	Extension name	Situation	Event nature	Description
21027	BPC.BFM.ACTIVITY.STATUS	Stop	TERMINATED	Activity terminated. Long-running activities can be terminated as an effect of fault handling on the scope or process the activity is assigned to.
21080	BPC.BFM.ACTIVITY.FAILURE	Failed	FAILED	Activity failed
21081	BPC.BFM.ACTIVITY.STATUS	Report	EXPIRED	Activity expired. You can define expiration on invoke and inline human task activities.
42005	BPC.BFM.ACTIVITY.STATUS	Report	SKIPPED	Activity skipped. This event can only apply to activities that have join behavior defined. If the join behavior evaluates to false, then the activity is skipped and the skipped event is emitted.
42012	BPC.BFM.ACTIVITY.MESSAGE	Report	OUTPUTSET	Activity output message set. Business object payload is available.
42013	BPC.BFM.ACTIVITY.MESSAGE	Report	FAULTSET	Activity fault message set. Business object payload is available.
42015	BPC.BFM.ACTIVITY.STATUS	Stop	STOPPED	Activity stopped
42031	BPC.BFM.ACTIVITY.STATUS	Report	FRETRIED	Activity forcibly retried. To force activities to retry use the Business Process Choreographer Explorer.
42032	BPC.BFM.ACTIVITY.STATUS	Stop	FCOMPLETED	Activity forcibly completed. To force activities to complete use the Business Process Choreographer Explorer.
42036	BPC.BFM.ACTIVITY.MESSAGE	Report	EXIT	Activity has message received

Code	Extension name	Situation	Event nature	Description
42037	BPC.BFM.ACTIVITY.STATUS	Report	CONDTRUE	Loop condition true
42038	BPC.BFM.ACTIVITY.STATUS	Report	CONDFALSE	Loop condition false
42039	BPC.BFM.ACTIVITY. WISTATUS	Report	WI_DELETED	Work item deleted. This event can only apply to pick, inline human tasks, and receive events.
42040	BPC.BFM.ACTIVITY. WISTATUS	Report	WI_CREATED	Work items created. This event can only apply to pick, inline human tasks, and receive events.
42050	BPC.BFM.ACTIVITY.ESCALATED	Report	ESCALATED	Activity escalated. This event can only apply to pick, inline human tasks, and receive events.
42054	BPC.BFM.ACTIVITY. WISTATUS	Report	WI_REFRESHED	Activity work items refreshed. This event can only apply to pick, inline human tasks, and receive events.
42055	BPC.BFM.ACTIVITY. WITRANSFER	Report	WI_TRANSFERRED	Work item transferred. This event can only apply to pick, inline human tasks, and receive events.
42057	BPC.BFM.ACTIVITY. FOREACH	Report	BRANCHES_STARTED	For each - activity branches started
42060	BPC.BFM.ACTIVITY. CUSTOMPROPERTYSET	Report	CP_SET	Activity custom property set. This event is generated when a custom property of an activity instance is changed.

For activity events, the following event correlation sphere identifiers have the following content:

- The *ECSCurrentID* provides the ID of the activity.
- The *ECSParentID* provides the ID of the containing process.

Activity scope events

The following table describes all activity scope events.

Code	Extension name	Situation	Event nature	Description
42020	BPC.BFM.ACTIVITY.STATUS	Start	ENTRY	Scope started
42021	BPC.BFM.ACTIVITY.STATUS	Report	SKIPPED	Scope skipped
42022	BPC.BFM.ACTIVITY.FAILURE	Fail	FAILED	Scope failed
42023	BPC.BFM.ACTIVITY.STATUS	Report	FAILING	Scope terminating
42024	BPC.BFM.ACTIVITY.STATUS	Stop	TERMINATED	Scope terminated
42026	BPC.BFM.ACTIVITY.STATUS	Stop	EXIT	Scope completed
42043	BPC.BFM.ACTIVITY.STATUS	Report	COMPENSATING	Scope compensating
42044	BPC.BFM.ACTIVITY.STATUS	Stop	COMPENSATED	Scope compensated
42045	BPC.BFM.ACTIVITY.STATUS	Fail	COMPFAILED	Scope compensation failed
42048	BPC.BFM.ACTIVITY.EVENT	Report	EV_RECEIVED	Activity event received
42051	BPC.BFM.ACTIVITY.ESCALATED	Report	EV_ESCALATED	Scope event escalated

Activity scope events are a type of activity events, whose syntax is described above for BPC.BFM.ACTIVITY.STATUS.

For activity scope events, the following event correlation sphere identifiers have the following content:

- The ECSCurrentID provides the ID of the scope.
- The ECSParentID provides the ID of the containing process.

Link events

The following table describes all link events.

Code	Extension name	Situation	Event nature	Description
21034	BPC.BFM.LINK.STATUS	Report	CONDTRUE	Link evaluated true
42000	BPC.BFM.LINK.STATUS	Report	CONDFALSE	Link evaluated false

For link events, the following event correlation sphere identifiers have the following content:

- The ECSCurrentID provides the ID of the source activity of the link.
- The ECSParentID provides the ID of the containing process.

Variable events

The following table describes the variable events.

Code	Extension name	Situation	Event nature	Description
21090	BPC.BFM.VARIABLE.STATUS	Report	CHANGED	Variable update. Business object payload is available.

For the variable event, the following event correlation sphere identifiers have the following content:

- The ECSCurrentID provides the ID of the containing process.
- The ECSParentID is the ECSCurrentID before the process instance start event of the current process.

Situations in business process events

Business process events can be emitted in different situations. The data for these situations is described in situation elements.

Business process events can contain one of the following situation elements.

Situation name	Content of the Common Base Event	
Start	categoryName is set to StartSituation.	
	situationType	
	Type	StartSituation
	reasoningScope	EXTERNAL
	successDisposition	SUCCESSFUL
	situationQualifier	START_COMPLETED
Stop	categoryName is set to StopSituation.	
	situationType	
	Type	StopSituation
	reasoningScope	EXTERNAL
	successDisposition	SUCCESSFUL
	situationQualifier	STOP_COMPLETED
Destroy	categoryName is set to DestroySituation.	
	situationType	
	Type	DestroySituation
	reasoningScope	EXTERNAL
	successDisposition	SUCCESSFUL
Fail	categoryName is set to StopSituation.	
	situationType	
	Type	StopSituation
	reasoningScope	EXTERNAL
	successDisposition	UNSUCCESSFUL
	situationQualifier	STOP_COMPLETED

Situation name	Content of the Common Base Event	
Report	categoryName is set to ReportSituation.	
	situationType	
	Type	ReportSituation
	reasoningScope	EXTERNAL
	reportCategory	STATUS

Monitoring human task events

Events that are emitted on behalf of human tasks consist of situation-independent data and data that is specific to human task events. The attributes and elements that are specific to human task events are described.

Human task events can have the following categories of event content.

Event data specific to human tasks

Events are created on behalf of tasks and escalations.

For Business Process Choreographer version 6.1 two event formats can occur:

WebSphere Business Monitor 6.0.2 format

WebSphere Business Monitor 6.0.2 format events occur when there are tasks modeled within WebSphere Integration Developer 6.0.2, or if the WebSphere Business Monitor 6.0.2 format mode is enabled within WebSphere Integration Developer 6.1. If not specified otherwise, the object-specific content for these events is written as *extendedDataElement* XML elements of the type string.

WebSphere Business Monitor 6.1 format

WebSphere Business Monitor 6.1 format events occur when there are tasks modeled within WebSphere Integration Developer 6.1, and the WebSphere Business Monitor 6.1 format mode is enabled. The object-specific content for these events is written as XML elements in the *xs:any* slot in the *eventPointData* folder of the Common Base Event. The structure of the XML is defined in the XML Schema Definition (XSD) file *HTMEvents.xsd*. The file can be found in the *install_root\ProcessChoreographer\client* directory.

Extension names for human task events

The extension name indicates the payload of the human task event. A list of all the extension names for human task events and their corresponding payload can be found here.

The extension name contains the string value that is used as the value of the *extensionName* attribute of the Common Base Event. This is also the name of the XML element that provides additional data about the event. The names of event elements are in uppercase, for example *BPC.HTM.BASE*, and the names of XML elements are in mixed case, for example, *HTMEventCode*. Except where indicated, all data elements are of the type string.

The following extension names are available for human task events:

- “BPC.HTM.BASE” on page 68
- “BPC.HTM.TASK.BASE” on page 68

- “BPC.HTM.TASK.STATUS”
- “BPC.HTM.TASK.FOLLOW”
- “BPC.HTM.TASK.MESSAGE” on page 69
- “BPC.HTM.TASK.INTERACT” on page 69
- “BPC.HTM.TASK.FAILURE” on page 69
- “BPC.HTM.TASK.WISTATUS” on page 69
- “BPC.HTM.TASK.WITRANSFER” on page 70
- “BPC.HTM.TASK.CUSTOMPROPERTYSET” on page 70
- “BPC.HTM.ESCALATION.BASE” on page 70
- “BPC.HTM.ESCALATION.STATUS” on page 70
- “BPC.HTM.ESCALATION.WISTATUS” on page 71
- “BPC.HTM.ESCALATION.WITRANSFER” on page 71
- “BPC.HTM.ESCALATION.CUSTOMPROPERTYSET” on page 71

BPC.HTM.BASE

BPC.HTM.BASE inherits the XML elements from WBIMonitoringEvent.

Table 44. XML elements for BPC.HTM.BASE

XML element	Description
<i>HTMEventCode</i>	The Business Process Choreographer event code that identifies the number of the event type. Possible event codes are listed in the following tables.
<i>taskTemplateId</i>	The ID of the template.
<i>taskTemplateName</i>	The name of the task template. This can differ from the display name.
<i>taskTemplateValidFrom</i>	The date and time from when the task template is valid.

BPC.HTM.TASK.BASE

BPC.HTM.TASK.BASE inherits the XML elements from “BPC.HTM.BASE.”

Table 45. XML elements for BPC.HTM.TASK.BASE

XML element	Description
<i>taskInstanceDescription</i>	The description of the task.

BPC.HTM.TASK.STATUS

BPC.HTM.TASK.STATUS inherits the XML elements from “BPC.HTM.TASK.BASE.”

BPC.HTM.TASK.FOLLOW

BPC.HTM.TASK.FOLLOW inherits the XML elements from “BPC.HTM.TASK.BASE.”

Table 46. XML elements for BPC.HTM.TASK.FOLLOW

XML element	Description
<i>followTaskId</i>	The ID of the task that was started as a follow-on task.

BPC.HTM.TASK.MESSAGE

BPC.HTM.TASK.MESSAGE inherits the XML elements from “BPC.HTM.TASK.BASE” on page 68.

Table 47. XML elements for BPC.HTM.TASK.MESSAGE

XML element	Description
<i>message</i> or <i>message_BO</i>	<p>A String or business object representation that contains the input or output message. The format depends on whether the Monitor Compatible Events option was selected on the Event Monitor tab in WebSphere Integration Developer.</p> <p>This attribute is only used for WebSphere Business Monitor 6.0.2 format events. For WebSphere Business Monitor 6.1 format events, the content of the message is written to the <i>applicationData</i> section, which contains one content element with the name set to the name of the message.</p>

BPC.HTM.TASK.INTERACT

BPC.HTM.TASK.INTERACT inherits the XML elements from “BPC.HTM.TASK.BASE” on page 68.

Table 48. XML elements for BPC.HTM.TASK.INTERACT

XML element	Description
<i>username</i>	The name of the user that is associated with the task.

BPC.HTM.TASK.FAILURE

BPC.HTM.TASK.FAILURE inherits the XML elements from “BPC.HTM.TASK.BASE” on page 68.

Table 49. XML elements for BPC.HTM.TASK.FAILURE

XML element	Description
<i>taskFailedException</i>	A string containing the <i>faultNameSpace</i> and <i>faultName</i> separated by a semicolon (;).

BPC.HTM.TASK.WISTATUS

BPC.HTM.TASK.WISTATUS inherits the XML elements from “BPC.HTM.TASK.BASE” on page 68.

Table 50. XML elements for BPC.HTM.TASK.WISTATUS

XML element	Description
<i>username</i>	The names of the users who have work items that were created or deleted.

BPC.HTM.TASK.WITRANSFER

BPC.HTM.TASK.WITRANSFER inherits the XML elements from “BPC.HTM.TASK.BASE” on page 68.

Table 51. XML elements for BPC.HTM.TASK.WITRANSFER

XML element	Description
<i>current</i>	The name of the current user. This is the user whose work item was transferred to someone else.
<i>target</i>	The name of the user of the work item receiver.

BPC.HTM.TASK.CUSTOMPROPERTYSET

BPC.HTM.TASK.CUSTOMPROPERTYSET inherits the XML elements from “BPC.HTM.TASK.BASE” on page 68.

Table 52. XML elements for BPC.HTM.TASK.CUSTOMPROPERTYSET

XML element	Description
<i>username</i>	The name of the user who set the custom property.
<i>propertyName</i>	The name of the custom property.
<i>propertyValue</i>	The value of the custom property.
<i>associatedObjectID</i>	The ID of the associated object that is the task instance ID.

BPC.HTM.ESCALATION.BASE

BPC.HTM.ESCALATION.BASE inherits the XML elements from “BPC.HTM.BASE” on page 68.

Table 53. XML elements for BPC.HTM.ESCALATION.BASE

XML element	Description
<i>escalationName</i>	The name of the escalation.
<i>escalationInstanceDescription</i>	The description of the escalation.

BPC.HTM.ESCALATION.STATUS

BPC.HTM.ESCALATION.STATUS inherits the XML elements from “BPC.HTM.ESCALATION.BASE.”

BPC.HTM.ESCALATION.WISTATUS

BPC.HTM.ESCALATION.WISTATUS inherits the XML elements from “BPC.HTM.ESCALATION.BASE” on page 70.

Table 54. XML elements for BPC.HTM.ESCALATION.WISTATUS

XML element	Description
<i>username</i>	The names of the users who have work items that are escalated.

BPC.HTM.ESCALATION.WITTRANSFER

BPC.HTM.ESCALATION.WITTRANSFER inherits the XML elements from “BPC.HTM.ESCALATION.BASE” on page 70.

Table 55. XML elements for BPC.HTM.ESCALATION.WITTRANSFER

XML element	Description
<i>current</i>	The name of the current user. This is the user whose work item was transferred to someone else.
<i>target</i>	The name of the user of the work item receiver.

BPC.HTM.ESCALATION.CUSTOMPROPERTYSET

BPC.HTM.ESCALATION.CUSTOMPROPERTYSET inherits the XML elements from “BPC.HTM.ESCALATION.BASE” on page 70.

Table 56. XML elements for BPC.HTM.ESCALATION.CUSTOMPROPERTYSET

XML element	Description
<i>username</i>	The name of the user who set the custom property.
<i>propertyName</i>	The name of the custom property.
<i>propertyValue</i>	The value of the custom property.
<i>associatedObjectID</i>	The ID of the associated object that is the escalation instance ID.

Human task events

Human task events are sent if monitoring is requested for the elements of the task in WebSphere Integration Developer. A list of all the events that can be emitted by human tasks can be found here.

An event is emitted when the state of a task changes. The following types of events can be caused by human tasks:

- “Task events” on page 72
- “Escalation events” on page 73

Note: Events are only emitted for ad-hoc tasks if the business relevance flag is set to true in the task model.

Events for inline tasks are emitted as activity events. For a list of these events, see “Business process events” on page 59.

XML Schema Definition (XSD) files

The event structure is described in the XML Schema Definition (XSD) file `HTMEvents.xsd`. The file can be found in the `install_root\ProcessChoreographer\client` directory.

Key to table columns

The columns in the following tables contain:

Code Contains the number of the event. For WebSphere Business Monitor 6.0.2 format events, the value is written to the Common Base Event as an extended data element with the name `HTMEventCode`. For WebSphere Business Monitor 6.1 format events, the value is written to the `xs:any` slot of the Common Base Event.

Extension name

Contains the string value that is used as the value of the `extensionName` attribute of the Common Base Event.

If WebSphere Business Integration Modeler is used to create the underlying task model, the extension name for events that contain message data in their payload can be extended by a hash character (#) followed by additional characters. These additional characters are used to distinguish Common Base Events that carry different message objects. Events that emit message data also contain additional nested `extendedDataElements` in order to report the contents of the data object. Refer to the documentation for WebSphere Business Integration Modeler for more information.

Situation

Refers to the situation name of the human task event. For details of situations, see “Situations in human task events” on page 74.

Event nature

A pointer to the event situation for a business process element in the `EventNature` parameter, as they are displayed in WebSphere Integration Developer.

Task events

The following table describes all task events.

Code	Extension name	Situation	Event nature	Description
51001	BPC.HTM.TASK.INTERACT	Report	CREATED	Task created
51002	BPC.HTM.TASK.STATUS	Destroy	DELETED	Task deleted
51003	BPC.HTM.TASK.STATUS	Start	ENTRY	Task started
51004	BPC.HTM.TASK.STATUS	Stop	EXIT	Task completed
51005	BPC.HTM.TASK.STATUS	Report	DEASSIGNED	Claim canceled
51006	BPC.HTM.TASK.INTERACT	Report	ASSIGNED	Task claimed
51007	BPC.HTM.TASK.STATUS	Stop	TERMINATED	Task terminated
51008	BPC.HTM.TASK.FAILURE	Fail	FAILED	Task failed
51009	BPC.HTM.TASK.STATUS	Report	EXPIRED	Task expired

Code	Extension name	Situation	Event nature	Description
51010	BPC.HTM.TASK.STATUS	Report	WAITFORSUBTASK	Waiting for subtasks
51011	BPC.HTM.TASK.STATUS	Stop	SUBTASKCOMPLETED	Subtasks completed
51012	BPC.HTM.TASK.STATUS	Report	RESTARTED	Task restarted
51013	BPC.HTM.TASK.STATUS	Report	SUSPENDED	Task suspended
51014	BPC.HTM.TASK.STATUS	Report	RESUMED	Task resumed
51015	BPC.HTM.TASK.FOLLOW	Report	COMPLETEDFOLLOW	Task completed and follow-on task started
51101	BPC.HTM.TASK.STATUS	Report	UPDATED	Task properties updated
51103	BPC.HTM.TASK.MESSAGE	Report	OUTPUTSET	Output message updated. Business object payload is available.
51104	BPC.HTM.TASK.MESSAGE	Report	FAULTSET	Fault message updated. Business object payload is available.
51201	BPC.HTM.TASK.WISTATUS	Destroy	WI_DELETED	Work item deleted
51202	BPC.HTM.TASK.WISTATUS	Report	WI_CREATED	Work items created
51204	BPC.HTM.TASK.WITRANSFER	Report	WI_TRANSFERRERD	Work item transferred
51205	BPC.HTM.TASK.WISTATUS	Report	WI_REFRESHED	Work items refreshed
51301	BPC.HTM.TASK.CUSTOMPROPERTYSET	Report	CP_SET	Custom property set. This event is generated when a custom property of a task instance is changed.

For task events, the following identifiers of event correlation spheres have the following content:

- The ESCcurrentID provides the ID of the task instance.
- The ECSParentID is the ECSCurrentID before the task instance event.

Escalation events

The following table describes all task escalation events.

Code	Extension name	Situation	Event nature	Description
53001	BPC.HTM.ESCALATION.STATUS	Report	ENTRY	Escalation fired
53201	BPC.HTM.ESCALATION.WISTATUS	Destroy	WI_DELETED	Work item deleted
53202	BPC.HTM.ESCALATION.WISTATUS	Report	WI_CREATED	Work item created
53204	BPC.HTM.ESCALATION.WITRANSFER	Report	WI_TRANSFERRERD	Escalation transferred
53205	BPC.HTM.ESCALATION.WISTATUS	Report	WI_REFRESHED	Work item refreshed

Code	Extension name	Situation	Event nature	Description
51302	BPC.HTM.ESCALATION. CUSTOMPROPERTYSET	Report	CP_SET	Custom property set. This event is generated when a custom property of an escalation instance is changed.

For task events, the following identifiers of event correlation spheres have the following content:

- The ESCcurrentID provides the ID of the escalation.
- The ECSParentID provides the ID of the associated task instance.

Situations in human task events

Human task events can be emitted in different situations. The data for these situations are described in situation elements.

Human task events can contain one of the following situation elements.

Situation name	Content of the Common Base Event	
Start	categoryName is set to StartSituation.	
	situationType	
	Type	StartSituation
	reasoningScope	EXTERNAL
	successDisposition	SUCCESSFUL
	situationQualifier	START_COMPLETED
Stop	categoryName is set to StopSituation.	
	situationType	
	Type	StopSituation
	reasoningScope	EXTERNAL
	successDisposition	SUCCESSFUL
	situationQualifier	STOP_COMPLETED
Destroy	categoryName is set to DestroySituation.	
	situationType	
	Type	DestroySituation
	reasoningScope	EXTERNAL
	successDisposition	SUCCESSFUL
Fail	categoryName is set to StopSituation.	
	situationType	
	Type	StopSituation
	reasoningScope	EXTERNAL
	successDisposition	UNSUCCESSFUL
	situationQualifier	STOP_COMPLETED

Situation name	Content of the Common Base Event	
Report	categoryName is set to ReportSituation.	
	situationType	
	Type	ReportSituation
	reasoningScope	EXTERNAL
	reportCategory	STATUS

Process server events

WebSphere Process Server features its own service components, and each of these components has its own set of event points that can be monitored. Service components contain one or more elements, which are sets of different steps processed in each service component. In turn, each element has its own set of event natures, that are key points that are reached when processing a service component element. All service components, their elements and associated event natures, and the extended data elements unique to each event are listed.

Resource Adapter events

The elements of the resource adapter component (base name eis:WBI.JCAAdapter) that can be monitored are listed here, along with their associated event natures, event names, and the extended data elements that are unique to each event.

Event Name	Event Natures	Event Contents	Type
InboundEventRetrieval element			
eis:WBI.JCAAdapter. InboundEventRetrieval. ENTRY	ENTRY	pollQuantity	int
		status	int
		eventTypeFilters	string
eis:WBI.JCAAdapter. InboundEventRetrieval. EXIT	EXIT	N/A	
eis:WBI.JCAAdapter. InboundEventRetrieval. FAILURE	FAILURE	FailureReason	exception
InboundEventDelivery element			
eis:WBI.JCAAdapter. InboundEventDelivery.ENTRY	ENTRY	N/A	
eis:WBI.JCAAdapter. InboundEventDelivery.EXIT	EXIT	N/A	
eis:WBI.JCAAdapter. InboundEventDelivery.FAILURE	FAILURE	FailureReason	exception
Outbound element			
eis:WBI.JCAAdapter. Outbound.ENTRY	ENTRY	N/A	
eis:WBI.JCAAdapter. Outbound.EXIT	EXIT	N/A	
eis:WBI.JCAAdapter. Outbound.FAILURE	FAILURE	FailureReason	exception
InboundCallbackAsyncDeliverEvent element			

Event Name	Event Natures	Event Contents	Type
eis:WBI.JCAAdapter. InboundCallbackAsyncDeliverEvent. ENTRY	ENTRY	N/A	
eis:WBI.JCAAdapter. InboundCallbackAsyncDeliverEvent. EXIT	EXIT	N/A	
eis:WBI.JCAAdapter. InboundCallbackAsyncDeliverEvent. FAILURE	FAILURE	FailureReason	exception
InboundCallbackSyncDeliverEvent element			
eis:WBI.JCAAdapter. InboundCallbackSyncDeliverEvent. ENTRY	ENTRY	N/A	
eis:WBI.JCAAdapter. InboundCallbackSyncDeliverEvent. EXIT	EXIT	N/A	
eis:WBI.JCAAdapter. InboundCallbackSyncDeliverEvent. FAILURE	FAILURE	FailureReason	exception
Polling element			
eis:WBI.JCAAdapter. Polling.STARTED	STARTED	PollFrequency	int
		PollQuantity	int
eis:WBI.JCAAdapter. Polling.STOPPED	STOPPED	N/A	
Delivery element			
eis:WBI.JCAAdapter. Delivery.EXIT	EXIT	N/A	
eis:WBI.JCAAdapter. Delivery.FAILURE	FAILURE	EventID	string
		FailureReason	exception
Retrieval element			
eis:WBI.JCAAdapter. Retrieval.FAILURE	FAILURE	EventID	string
		FailureReason	exception
Endpoint element			
eis:WBI.JCAAdapter. Endpoint.FAILURE	FAILURE	FailureReason	exception
Recovery element			
eis:WBI.JCAAdapter. Recovery.EXIT	EXIT	N/A	
eis:WBI.JCAAdapter. Recovery.FAILURE	FAILURE	FailureReason	exception
EventFailure element			
eis:WBI.JCAAdapter. EventFailure.FAILURE	FAILURE	FailureReason	exception
Connection element			
eis:WBI.JCAAdapter. Connection.FAILURE	FAILURE	FailureReason	exception

Business rule events

The business rule component (base name br:WBI.BR) contains a single element that can be monitored. All event types for this element are listed here, with their associated event natures, event names, and the extended data elements that are unique to each event.

Event Name	Event Nature	Event Contents	Type
br:WBI.BR.ENTRY	ENTRY	operationName	string
br:WBI.BR.EXIT	EXIT	operationName	string
br:WBI.BR.FAILURE	FAILURE	ErrorReport	Exception
		operationName	string
WBI.BR. br:SelectionKeyExtracted	SelectionKeyExtracted	operationName	string
br:WBI.BR.TargetFound	TargetFound	operationName	string
		target	string

Business state machine events

The elements from the business state machine component (base name bsm:WBI.BSM) that can be monitored are listed here, along with their associated event natures, event names, and all extended data elements that are unique to each event.

Event Name	Event Nature	Event Contents	Type
StateMachineDefinition element			
bsm:WBI.BSM. StateMachineDefinition. ALLOCATED	ALLOCATED	instanceID	string
bsm:WBI.BSM. StateMachineDefinition. RELEASED	RELEASED	instanceID	string
Transition element			
bsm:WBI.BSM.Transition.ENTRY	ENTRY	instanceID	string
		name	string
bsm:WBI.BSM.Transition.EXIT	EXIT	instanceID	string
		name	string
bsm:WBI.BSM.Transition.FAILURE	FAILURE	ErrorReport	Exception
		instanceID	string
		name	string
State element			
bsm:WBI.BSM.State.ENTRY	ENTRY	instanceID	string
		name	string
bsm:WBI.BSM.State.EXIT	EXIT	instanceID	string
		name	string
bsm:WBI.BSM.State.FAILURE	FAILURE	ErrorReport	Exception
		instanceID	string
		name	string

Event Name	Event Nature	Event Contents	Type
Guard element			
bsm:WBI.BSM.Guard.ENTRY	ENTRY	instanceID	string
		name	string
bsm:WBI.BSM.Guard.EXIT	EXIT	instanceID	string
		name	string
		result	boolean
bsm:WBI.BSM.Guard.FAILURE	FAILURE	ErrorReport	Exception
		instanceID	string
		name	string
Action element			
bsm:WBI.BSM.Action.ENTRY	ENTRY	instanceID	string
		name	string
bsm:WBI.BSM.Action.EXIT	EXIT	instanceID	string
		name	string
bsm:WBI.BSM.Action.FAILURE	FAILURE	ErrorReport	Exception
		instanceID	string
		name	string
EntryAction element			
bsm:WBI.BSM.EntryAction. ENTRY	ENTRY	instanceID	string
		name	string
bsm:WBI.BSM.EntryAction. EXIT	EXIT	instanceID	string
		name	string
bsm:WBI.BSM.EntryAction. FAILURE	FAILURE	ErrorReport	Exception
		instanceID	string
		name	string
ExitAction element			
bsm:WBI.BSM.ExitAction.ENTRY	ENTRY	instanceID	string
		name	string
bsm:WBI.BSM.ExitAction.EXIT	EXIT	instanceID	string
		name	string
bsm:WBI.BSM.ExitAction. FAILURE	FAILURE	ErrorReport	Exception
		instanceID	string
		name	string
Timer element			
bsm:WBI.BSM.Timer.START	START	instanceID	string
		name	string
		duration	string
bsm:WBI.BSM.Timer.STOPPED	STOPPED	instanceID	string
		name	string
		duration	string

Map events

The elements from the map component (base name `map:WBI.MAP`) that can be monitored are listed here, along with their event natures, event names, and all extended data elements that are unique to each event.

Table 57. Base element

Event Name	Event Nature	Event Contents	Type
<code>map:WBI.MAP.ENTRY</code>	ENTRY	N/A	N/A
<code>map:WBI.MAP.EXIT</code>	EXIT	N/A	N/A
<code>map:WBI.MAP.FAILURE</code>	FAILURE	FailureReason	Exception
Transformation element			
<code>map:WBI.MAP.Transformation. ENTRY</code>	ENTRY	N/A	N/A
<code>map:WBI.MAP.Transformation. EXIT</code>	EXIT	N/A	N/A
<code>map:WBI.MAP.Transformation. FAILURE</code>	FAILURE	FailureReason	Exception

Mediation events

The elements from the mediation component (base name `ifm:WBI.MEDIATION`) that can be monitored are listed here, along with their associated event natures, names, and all extended data elements that are unique to each event.

Event Name	Event Nature	Event Contents	Type
OperationBinding element			
<code>ifm:WBI.MEDIATION. OperationBinding.ENTRY</code>	ENTRY	InteractionType	string
		TicketID	string
		Source	string
		Target	string
<code>ifm:WBI.MEDIATION. OperationBinding.EXIT</code>	EXIT	InteractionType	string
		TicketID	string
		Source	string
		Target	string
<code>ifm:WBI.MEDIATION. OperationBinding.FAILURE</code>	FAILURE	InteractionType	string
		TicketID	string
		Source	string
		Target	string
		ErrorReport	Exception
ParameterMediation element			
<code>ifm:WBI.MEDIATION. ParameterMediation. ENTRY</code>	ENTRY	Type	string
		TransformName	string
<code>WBI.MEDIATION. ParameterMediation. EXIT</code>	EXIT	Type	string
		TransformName	string

Event Name	Event Nature	Event Contents	Type
ifm:WBI.MEDIATION. ParameterMediation. FAILURE	FAILURE	Type	string
		TransformName	string
		ErrorReport	Exception

Recovery events

The recovery component (base name `recovery:WBI.Recovery`) contains a single element that can be monitored. All event types for this element are listed here, along with their associated event natures, event names, and the extended data elements that are unique to each event.

Event Name	Event Nature	Event Contents	Type
recovery:WBI.Recovery. FAILURE	FAILURE	MsgId	string
		DestModuleName	string
		DestComponentName	string
		DestMethodName	string
		SourceModuleName	string
		SourceComponentName	string
		ResubmitDestination	string
		ExceptionDetails	string
		SessionId	string
		FailureTime	dateTime
		ExpirationTime	dateTime
		Status	int
		MessageBody	byteArray
Deliverable	boolean		
recovery:WBI.Recovery. DEADLOOP	DEADLOOP	DeadloopMsgId	string
		SIBusName	string
		QueueName	string
		Reason	string
recovery:WBI.Recovery. RESUBMIT	RESUBMIT	MsgId	string
		OriginalMesId	string
		ResubmitCount	int
		Description	string
recovery:WBI.Recovery. DELETE	DELETE	MsgId	string
		deleteTime	dateTime
		Description	string

Service Component Architecture events

The Service Component Architecture (SCA) contains a single element, with a base name of `sca:WBI.SCA.MethodInvocation`. All the events and associated natures of this element are listed here, along with all extended data elements and that are unique to each event.

Note: These events should not be confused with the SCA-specific ARM performance statistics.

Event Name	Event Nature	Event Contents	Type
WBI.SCA. MethodInvocation. ENTRY	ENTRY	SOURCE COMPONENT	string
		SOURCE INTERFACE	string
		SOURCE METHOD	string
		SOURCE MODULE	string
		SOURCE REFERENCE	string
		TARGET COMPONENT	string
		TARGET INTERFACE	string
		TARGET METHOD	string
WBI.SCA. MethodInvocation. EXIT	EXIT	SOURCE COMPONENT	string
		SOURCE INTERFACE	string
		SOURCE METHOD	string
		SOURCE MODULE	string
		SOURCE REFERENCE	string
		TARGET COMPONENT	string
		TARGET INTERFACE	string
		TARGET METHOD	string
WBI.SCA. MethodInvocation. FAILURE	FAILURE	SOURCE COMPONENT	string
		SOURCE INTERFACE	string
		SOURCE METHOD	string
		SOURCE MODULE	string
		SOURCE REFERENCE	string
		TARGET COMPONENT	string
		TARGET INTERFACE	string
		TARGET METHOD	string
		TARGET MODULE	string
		Exception	string

Selector events

The selector component contains a single element that can be monitored. All event types for this element are listed here, along with their associated event natures, event names, and the extended data elements that are unique to each event. All

selector events have a base name of sel:WBI.SEL.

Event Name	Event Nature	Event Contents	Type
sel:WBI.SEL.ENTRY	ENTRY	operationName	string
sel:WBI.SEL.EXIT	EXIT	operationName	string
sel:WBI.SEL.FAILURE	FAILURE	ErrorReport	Exception
		operationName	string
sel:WBI.SEL. SelectionKeyExtracted	SelectionKeyExtracted	operationName	string
sel:WBI.SEL.TargetFound	TargetFound	operationName	string
		target	string

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

*IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.*

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

*IBM World Trade Asia Corporation Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan*

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
577 Airport Blvd., Suite 800
Burlingame, CA 94010
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows: (c) (your company name) (year). Portions of

this code are derived from IBM Corp. Sample Programs. (c) Copyright IBM Corp. _enter the year or years_. All rights reserved.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Programming interface information

Programming interface information, if provided, is intended to help you create application software using this program.

General-use programming interfaces allow you to write application software that obtain the services of this program's tools.

However, this information may also contain diagnosis, modification, and tuning information. Diagnosis, modification and tuning information is provided to help you debug your application software.

Warning: Do not use this diagnosis, modification, and tuning information as a programming interface because it is subject to change.

Trademarks and service marks

IBM, the IBM logo, developerWorks, Tivoli, and WebSphere are registered trademarks of International Business Machines Corporation in the United States, other countries, or both.

Adobe is a registered trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

This product includes software developed by the Eclipse Project (<http://www.eclipse.org>).



IBM WebSphere Process Server for Multiplatforms, Version 6.1.0



Printed in USA