

IBM WebSphere Process Server for Multiplatforms



Monitoring WebSphere Process Server

Version 7.0.0

30 April 2010

This edition applies to version 7, release 0, 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 IBM Corporation 2006, 2010.**

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

Contents

Service component monitoring overview 1

Why use monitoring?	1
What do you monitor?	2
How do you enable monitoring?.	3

Enabling and configuring service component monitoring 7

Monitoring performance	7
Performance Monitoring Infrastructure statistics	8
Application Response Measurement statistics for the Service Component Architecture	13
Monitoring service component events	27
Enabling monitoring of business process and human task events	27
Configuring logging for service component events	28
Monitoring service components with the Common Event Infrastructure server	34
Session monitoring	38

Viewing monitored events 41

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

Event catalog 45

The Common Base Event standard elements	45
Business objects in events.	46
Business Process Choreographer events	47
WebSphere Process Server events	47
Resource Adapter events	47
Business rule events	49
Business state machine events	49
Map events	51
Mediation events	51
Recovery events	52
Service Component Architecture events	53
Selector events	54

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 designed for monitoring process server service components. This section focuses on how you monitor server-specific service components. It is intended to supplement the monitoring and troubleshooting topics found in the WebSphere Application Server Information Center; therefore, refer to that documentation for details of the other monitoring capabilities in the combined product.

Why use monitoring?

You monitor service components within WebSphere Process Server to assess performance, to troubleshoot problems, and to 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 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 that the server is intended to accomplish. There are three main reasons you need to monitor service components on the 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 make up 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 generate (or “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 monitor a certain point that is reached during 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 these factors determine the type of event generated by monitoring.

Event natures describe the situations required to generate events during the processing of service components. These natures are 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 later 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 activity is detected, 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:

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 server-specific server components:

- A counter for each EXIT event nature – counts successful computations.
- A counter for each FAILURE event nature – 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 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.

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.

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. In

both cases, the events generated by monitoring are 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 events 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 application. When WebSphere Process Server has been configured to use a CEI server, the monitoring function begins firing service component events to a CEI server whenever the specified services are invoked. As long as the application is deployed on WebSphere Process Server, the service component event points specified in the .mon file is 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 while 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 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 need to use the CEI server database to access the service component events related to that session.

Related concepts

“Monitoring performance” on page 7

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.

“Session monitoring” on page 38


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.


Related tasks

“Enabling and configuring service component monitoring” on page 7

To be able to monitor service components, 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.

 [Administering Common Event Infrastructure](#)

 [Enabling Common Base Events and the audit trail, using the administrative console](#)

 [Getting performance data from request metrics](#)

Enabling and configuring service component monitoring

To be able to monitor service components, 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.

Related concepts

“How do you enable monitoring?” on page 3

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.

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 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 events that include 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 making up an enterprise application. You can derive various performance measurements for the processing of your applications using these statistics.

Related concepts

“How do you enable monitoring?” on page 3

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.

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 > Server Types > WebSphere 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 updates.
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 use performance monitoring statistics to monitor the counts of successful and unsuccessful invocation requests, and the time taken to complete events.

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

Related reference

“Application Response Measurement statistics for the Service Component Architecture” on page 13

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

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 the statistics that can be monitored are in the listed groups. Some statistics cannot be listed because they have not been invoked since the server was last started.
5. Select a statistic you want to monitor from the tree on the left side of the panel, and then select the statistics that you want to collect on the right side, then click **Enable**. Repeat 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, Do 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 can cause one or the other type of statistic to appear skewed in the viewer graph.

Tutorial: Service component performance monitoring

This tutorial guides you through an example of setting up performance monitoring, and how to view the resulting statistics.

For service component event points that you monitor, you can publish to the Performance Monitoring Infrastructure (PMI) and view the resulting performance statistics on the Tivoli Performance Viewer (TPV). This exercise demonstrates 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 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 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 performance monitoring function in the server fires "minimized" events that do not contain all 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 server.
- Enabled the PMI on the server.
- Installed and started the Samples Gallery application on the server.
- Installed and started the business rules sample application on the server. Follow the instructions on the Samples Gallery page to set up and run the business rules sample application.

After all 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:

For monitoring performance, you can use the administrative console to select service components for monitoring and view performance measurements. This example shows the use of the console to monitor performance statistics.

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, because you will be running the sample several times after you begin monitoring. Ensure that you have already run the sample at least once, which causes it to appear in the list of functions that you can select to monitor.

Procedure

1. Open the administrative console.
2. Select the cluster or server to monitor.
 - To monitor a cluster, click **Servers** → **Clusters** → **WebSphere application server clusters** → *cluster_name*.
 - To monitor a single server, click **Servers** → **Server Types** → **WebSphere application servers** → *server_name*.
3. Click the Runtime tab.
4. Under Performance, click **Performance Monitoring Infrastructure**.
5. Select **Custom**.
6. Expand **WBISStats.RootGroup** → **BR** → **brsample_module.DiscountRuleGroup** → **Operation**.
7. Select **_calculateDiscount**
8. Select the check boxes next to **BadRequests**, **GoodRequests**, and **ResponseTime**.
9. Click **Enable**
10. In the navigation pane, click **Monitoring and Tuning** → **Performance Viewer** → **Current Activity**.
11. Select the check box next to *server_name*, then click **Start Monitoring**.
12. Click *server_name*.
13. Expand **WBISStats.RootGroup** → **BR** → **brsample_module.DiscountRuleGroup** → **Operation**.
14. 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 Tivoli Performance Viewer 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.

After you have completed this example, you should understand how WebSphere Process Server implements performance monitoring of service components. You should 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.

What to do next

Performance monitoring can tax system resources; therefore, after you have completed this task you should stop the monitors. To do this, click 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 (SCA) level. You can use these Application Response Measurement (ARM) statistics, which are either counters or timers, to measure invocations to and responses from services in various patterns.

The Application Response Measurement (ARM) statistics shown in the following tables are — in a 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 patterns, 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 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:

- `Arm40.ArmMetricFactory` — the full Java class name of your ARM implementation providers metrics factory.
- `Arm40.ArmTranReportFactory` — the full Java class name of your ARM implementation providers transaction report factory.
- `Arm40.ArmTransactionFactory` — the full Java class name of your ARM implementation providers 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
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.
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. For a deferred response, this time does not 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.
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.

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

Statistic name	Type	Description
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.

Related reference

"Performance Monitoring Infrastructure statistics" on page 8

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.

Related information



 [WebSphere Application Server documentation Network Deployment](#)

Synchronous invocations

You can obtain Application Response Measurement (ARM) performance statistics from a simple Service Component Architecture (SCA) call to a service and the response from the service.

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.

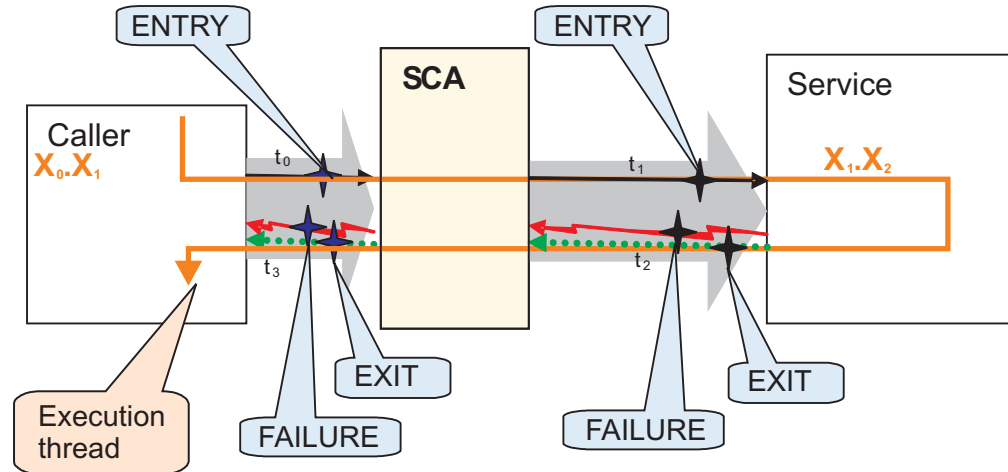
In the table and diagram below, the "current" ARM transaction (denoted as X_1) is created when the calling service component was invoked for the first time. If the caller is not a service component, the current ARM transaction is used, or a new one is created. If it is not the starting transaction then it has a parent, as represented in the following table and diagram with the notation $X_n.X_{n+1}$. The notation is used to document 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 they do 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	Count _{EXIT}	

Table 8. ARM statistics for synchronous invocations of SCA (continued)

Statistics	Formula	ARM Transaction
BadRequests	$\text{Count}_{\text{FAILURE}}$	
ProcessTime	$t_2 - t_1$	





Deferred response with synchronous implementation

You can obtain Application Response Measurement (ARM) statistics with a synchronous invocation of the request. The returned result is sent as output to a data store for a synchronous implementation.

Parameters

Event monitoring for Service Component Architecture (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 is used, or a new one is created. If it is not the starting transaction, it has a parent, as represented in the following table and diagram with the notation $X_n.X_{n+1}$. The notation is 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 you cannot 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	$t_1 - t_0$	

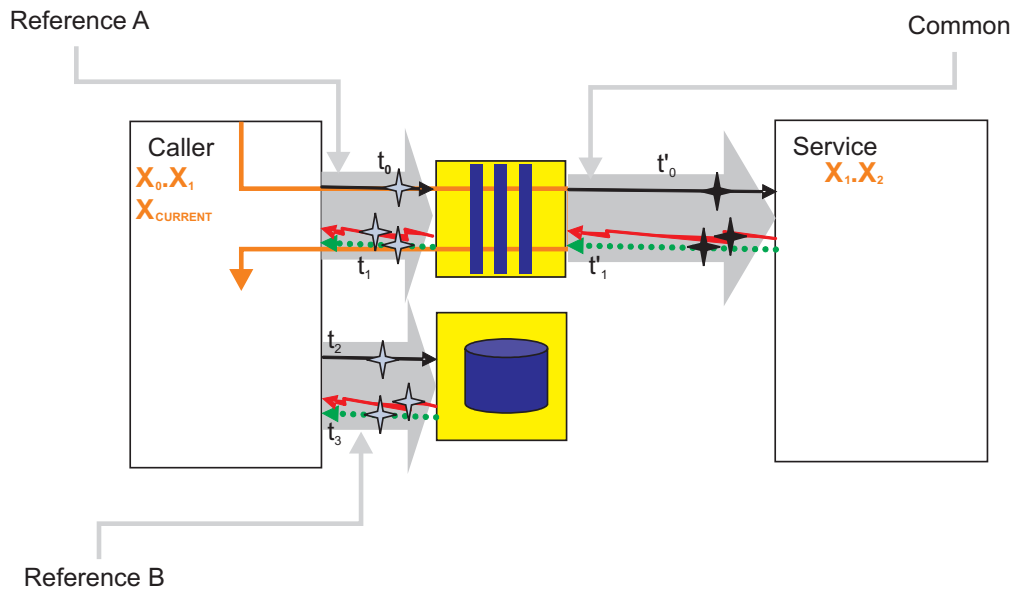


Table 10. Invocation of output to data source

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

You can obtain Application Response Measurement (ARM) statistics from an asynchronous implementation. The call to the service and the return result are invoked but the resulting output is sent to a data store from the service target.

Parameters

Event monitoring for Service Component Architecture (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 is used, or a new one is created. If it is not the starting transaction, it has a parent, as represented in the following table and diagram with the notation $X_n \cdot X_{n+1}$. The notation is 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 you cannot modify the SCA transaction lineage.

Table 11. Invocation of request and return result

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

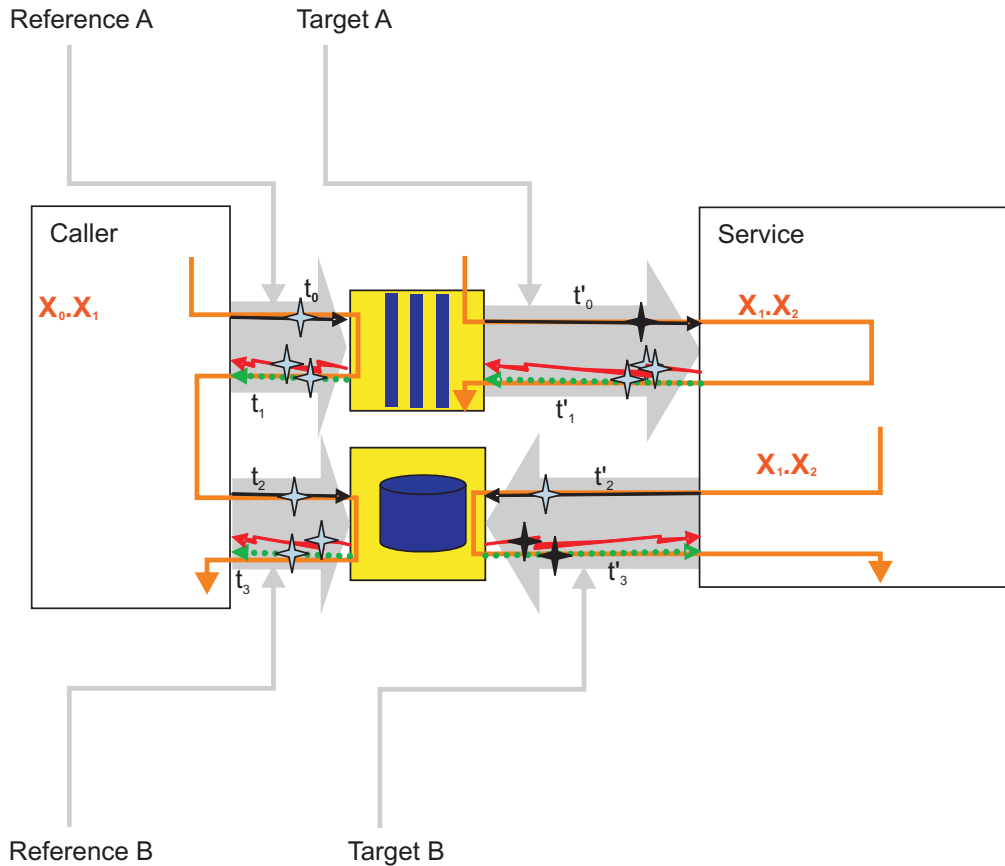


Table 12. Invocation of return result to a data store



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

Deferred response with asynchronous result retrieve

The ResultRetrieve Application Response Measurement (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 Service Component Architecture (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 one 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	$t_1 - t_0$	

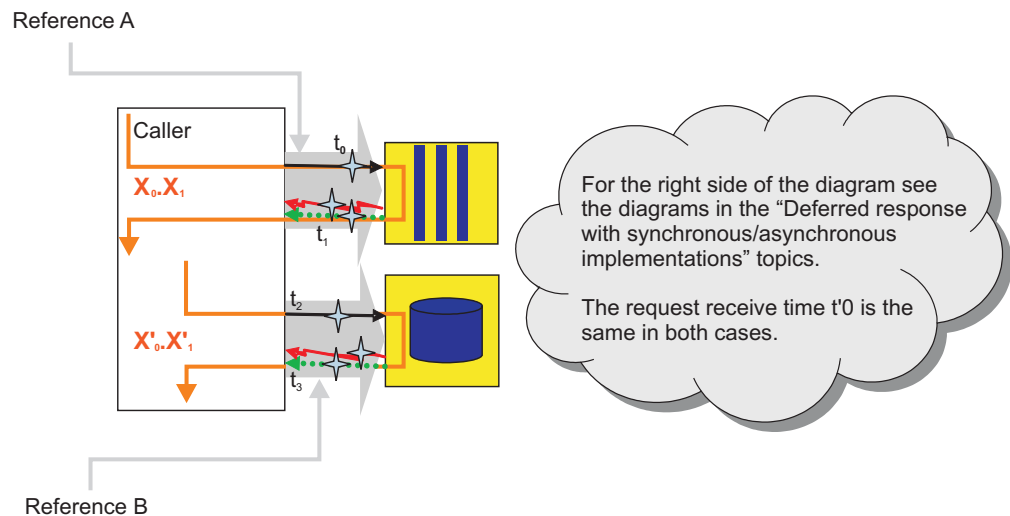


Table 14. Invocation of request and return result



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

Asynchronous callback with synchronous implementation

You can obtain Application Response Measurement (ARM) statistics when callback requests and callback executions use different threads on a synchronous implementation.

Parameters

Event monitoring for Service Component Architecture (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 is used, or a new one is created. If it is not the starting transaction, it has a parent, as represented in the following table and diagram with the notation $X_n \cdot X_{n+1}$. The notation is 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 you cannot 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	$t'_0 - t_0$	
	ResponseDeliveryTime	$t_2 - t'_1$	
	GoodRequests	Count _{EXIT}	
	BadRequests	Count _{FAILURE}	
	ResponseTime	$t_3 - t_2$	
Reference	GoodRefRequest	Count _{EXIT}	$X_1 \cdot X_2$
	BadRefRequests	Count _{FAILURE}	
	RefResponseTime	$t'_1 - t'_0$	

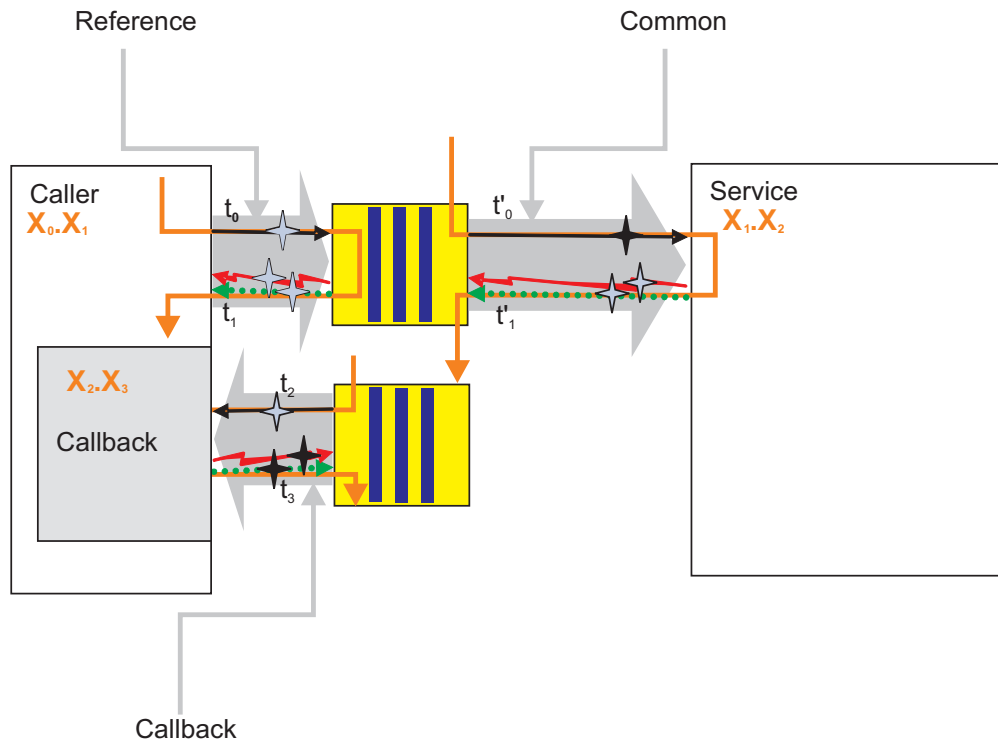


Table 16. Invocation of callback

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

Application Response Measurement (ARM) statistics are available for callback requests and callback executions using different threads with an asynchronous implementation

Parameters

Event monitoring for Service Component Architecture (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 is used, or a new one is created. If it is not the starting transaction it has a parent, as represented in the following table and diagram with the notation $X_n.X_{n+1}$. The notation is 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 you cannot modify the SCA transaction lineage.

Table 17. Invocation of request and return result

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

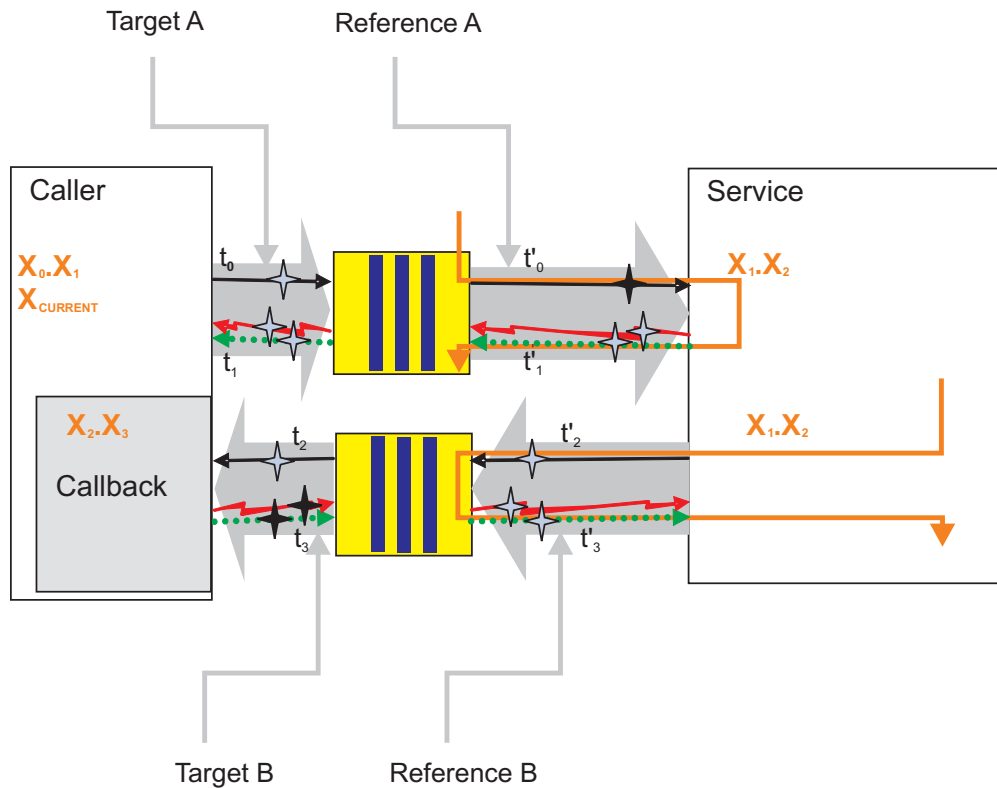


Table 18. Invocation of callback



Type	Statistics	Formula	ARM Transaction
Reference B	GoodCBSubmit	Count _{EXIT}	$X_1 \cdot X_2$
	BadCBSubmit	Count _{FAILURE}	
	CBSubmitTime	$t'_3 - t'_2$	
Target B	GoodCB	Count _{EXIT}	$X_0 \cdot X_1$
	BadCB	Count _{FAILURE}	
	CBTime	$t_3 - t_2$	

Asynchronous one way with synchronous implementation

These Application Response Measurement (ARM) statistics can be obtained when a call is submitted (fire and forget) with a synchronous implementation.

Parameters

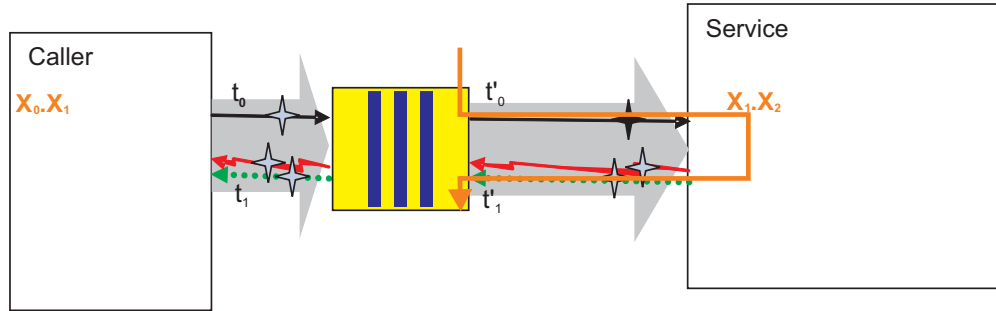
Event monitoring for Service Component Architecture (SCA) components includes

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 is used, or a new one is created. If it is not the starting transaction, it has a parent, as represented in the following table and diagram with the notation $X_n \cdot X_{n+1}$. The notation is 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 you cannot 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

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

Parameters

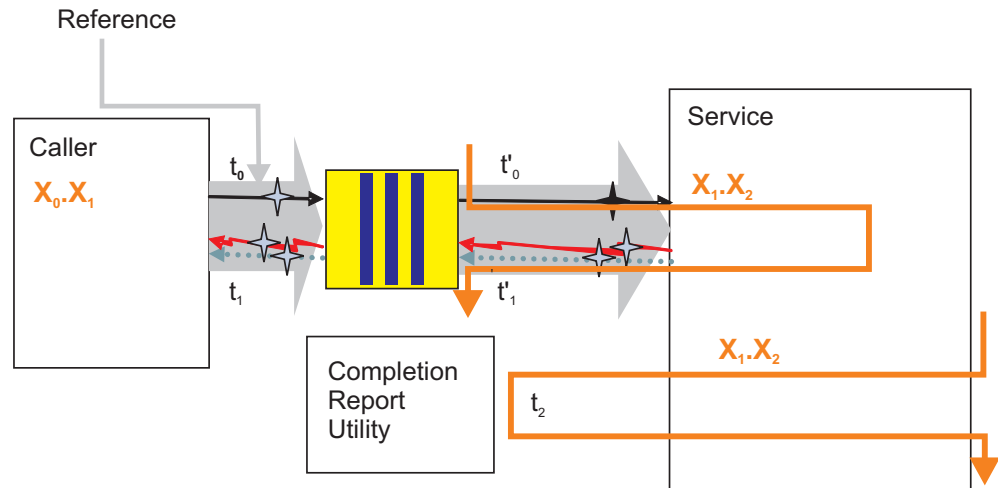
Event monitoring for Service Component Architecture (SCA) components includes

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 is used, or a new one is created. If it is not the starting transaction, it has a parent. This relationship is represented in the following table and diagram with the notation $X_n \cdot X_{n+1}$. The notation is 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 you cannot modify the SCA transaction lineage.

Table 20. Invocation of request and return result

Type	Statistics	Formula	ARM Transaction
Common	TotalResponseTime	$t_1 - t_0$	$X_0 \cdot X_1$
	RequestDeliveryTime	$t'_0 - t_0$	$X_1 \cdot X_2$
	ResponseDeliveryTime	N/A	N/A
	GoodRequests	$\text{Count}_{\text{EXIT}}$	$X_1 \cdot X_2$
	BadRequests	$\text{Count}_{\text{FAILURE}}$	
	ResponseTime	$t_2 - t_0$	
Reference	GoodRefRequest	$\text{Count}_{\text{EXIT}}$	$X_0 \cdot X_1$
	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. After 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. To enable business process events for the Human Task Manager, click **Servers** → **Server Types** → **WebSphere application servers** → *server_name*, then on the **Configuration** tab under **Business Integration**, expand **Business Process Choreographer**, click **Human Task Manager**. In the section **State Observers**, ensure that the boxes for **Enable Common Event Infrastructure Logging**, **Enable audit logging**, and **Enable task history** are selected. If the check boxes are not selected, then you must select them and restart the server.
 - b. To enable business process events for the Business Flow Manager, click **Servers** → **Server Types** → **WebSphere application servers** → *server_name*, then on the **Configuration** tab under **Business Integration**, expand **Business Process Choreographer**, click **Business Flow Manager**. In the section **State Observers**, ensure that the boxes for **Enable Common Event Infrastructure Logging** and **Enable audit logging** are selected. 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. To enable business process events for the Human Task Manager, click **Servers** → **Clusters** → **WebSphere application server clusters** → *cluster_name*, then on the **Configuration** tab under **Business Integration**, expand **Business Process Choreographer**, click ensure that the boxes for **Enable Common Event Infrastructure Logging**, **Enable audit logging**, and **Enable task history** are selected. If the check boxes are not selected, then you must select them and restart the server.
 - b. To enable business process events for the Business Flow Manager, click **Servers** → **Clusters** → **WebSphere application server clusters** → *cluster_name*, then on the **Configuration** tab under **Business Integration**, expand **Business Process Choreographer**, click **Business Flow Manager**. In the section **State Observers**, ensure that the boxes for **Enable Common Event Infrastructure Logging** and **Enable audit logging** are selected. If the check boxes are not selected, then you must select them and restart the server.

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 can 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 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** → **Server Types** → **WebSphere 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 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 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 > Server Types > WebSphere 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 server events that have been invoked at least once appear on this list. All 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 server.

Results

By default, the loggers 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

For service component event points that you monitor, events can be published to the logging facilities of the underlying WebSphere Application Server. This tutorial guides you through an example of setting up monitoring with logging, and how to view events stored in a 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 a 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 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 server.
- Configured Common Event Infrastructure.
- Enabled the diagnostic trace service on the server.
- Installed and started the Samples Gallery application on the server.
- Installed and started the business rules sample application on the 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:

For monitoring with logging, you can use the administrative console to manage the details for event types. This example shows the use of the console to change the level of detail recorded for some event types and to use a text editor to open the trace.log file to view the information for individual events.

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. To view the published events, open the log file in a text editor, and decipher 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 logs 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 can 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.
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.

Scenario	Result	Log entry contents
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 installation	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 (CEI) server. Service component event points can be specified for monitoring with the CEI server on a permanent basis for viewing and managing application flow, or on a temporary basis for troubleshooting problems.

You can use monitoring to publish the data in service component event points within service component events that are fired across the 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.

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 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 actions of, and interactions between, the service component processes running on your system. You can also quickly detect deviations from the normal flow of these processes, which can indicate that your service components are not working properly.

To configure static monitoring of service components, you use WebSphere Integration Developer to select the service component event points in your applications. The selections are specified in the form of an XML file with a .mon extension that is deployed along with the application. After you have deployed the application on a running server, you cannot turn off or lower the detail level of the monitoring for events specified in the .mon file of the application. To stop this monitoring, you must stop the server and undeploy the application.

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 the output that is published to loggers, with Common Base Event elements that make up 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 guides you through an example of setting up monitoring with the CEI server, and how to view events stored in the database.

For service component event points that you monitor, events can be published to the Common Event Infrastructure (CEI) server and stored in the CEI server database. Once events have been captured, use the Common Base Event browser to view those stored events. The example you use in this scenario does not involve static monitoring, whereby an application deployed with a .mon file continually monitors specific service components event points. For information about how to perform static monitoring, consult the IBM® WebSphere Integration Developer Information Center.

The scenario you follow for this example, instead, shows you how to select for monitoring event points on service components in applications already deployed and running on a server. You can 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 stores the event information about its database. You then 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 server.
- Configured the CEI and its database.
- Enabled the diagnostic trace service on the server.
- Installed and started the Samples Gallery application on the server.
- Installed and started the business rules sample application on the server. Follow the instructions on the Samples Gallery page to set up and run the business rules sample application.

After all 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:

For monitoring with the CEI server, you can use the administrative console to manage the details for event types and to display recorded events in the Common Base Event browser. This example shows the use of the console to change the level of detail recorded for some event types and to use the Common Base Event browser to view the information for individual events.

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** → **Server Types** → **WebSphere 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 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 can see columns of links for more details about events. 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 Choreographer Explorer and view further information about the business process or human task event.

Related concepts

“How do you enable monitoring?” on page 3

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.

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

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

Before you can view performance metrics with the Tivoli Performance Viewer, the following conditions must be true:

- The servers that you want to monitor must be running on the node
- The Performance Monitoring Infrastructure (PMI) is enabled
- 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 various details of about the performance of your server. The section entitled “Monitoring performance with Tivoli Performance Viewer” in the WebSphere Application Server Information Center contains details about how to use this tool for various purposes, including the resource for complete instructions on using this program. This section is limited to discussing the viewing of performance data for events specific to WebSphere Process Server.

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, do not mix counter-type statistics with duration-type statistics. Counters are cumulative, and the scales against which they are graphed 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 can cause one or the other type of statistic to appear skewed in the viewer graph.

Procedure

- View current performance activity
 1. Click **Monitoring and Tuning** → **Performance Viewer** → **Current Activity** in the administrative console navigation tree.

2. Select **Server**, then 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, then click **Start Monitoring**. To start monitoring multiple servers at the same time, select the servers then click **Start Monitoring**.
 3. Select **Performance Modules**.
 4. Select the check box beside the name of each performance module that you want to view. WebSphere Process Server events that emit performance statistics, and that have been invoked at least once, are listed under the WBIStats.RootGroup hierarchy. Expand the tree by clicking + next to a node and shrink it by clicking – next to a node.
 5. Click **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 then 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 **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.
 1. Click **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. 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 also includes several logger-specific fields. 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

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 like 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 can 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 can be some other identifier for user applications.

EventType

A one-character field that indicates the type of the trace event. Trace types are in lowercase. 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.

Event catalog

The event catalog contains the specifications for all the events that can be monitored for each service component type, and the associated Common Base Event extended data elements produced by each event.

Use the information presented in this section as reference material that enables you to understand how individual events are structured. This knowledge helps 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 following items:

- The structure and standard elements of the Common Base Event
- The list of events for the Business Process Choreographer service components
- The list of WebSphere Process Server-specific service components
- The extensions to the Common Base Event unique to each event type

There is also a discussion of how business objects that might 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, 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.

Attribute	Description
location	Set to the host name of the executing server.
application	Not used.
executionEnvironment	A string that identifies the operating system.
component	Process server version. For business processes and human tasks: Set to WPS#, followed by the SCA version, 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 specific components, set to ReportSituation.
Situation Category	The category of the type of situation that caused the event to be reported. For specific components, set to STATUS.
Situation Reasoning Scope	The scope of the impact of the situation reported. For 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 can 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.

WebSphere 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 event types available for the resource adapter component are listed.

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

Event Name	Event Natures	Event Contents	Type
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			
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			

Event Name	Event Natures	Event Contents	Type
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 event types available for the business rule component are listed.

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 event types available for the business state machine component are listed.

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

Event Name	Event Nature	Event Contents	Type
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
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

Event Name	Event Nature	Event Contents	Type
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 event types available for the map component are listed.

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 21. Base element

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

Mediation events

The event types available for the mediation component are listed.

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			
ifm:WBI.MEDIATION. OperationBinding.ENTRY	ENTRY	InteractionType	string
		TicketID	string
		Source	string
		Target	string
ifm:WBI.MEDIATION. OperationBinding.EXIT	EXIT	InteractionType	string
		TicketID	string
		Source	string
		Target	string
ifm:WBI.MEDIATION. OperationBinding.FAILURE	FAILURE	InteractionType	string
		TicketID	string
		Source	string
		Target	string
		ErrorReport	Exception
ParameterMediation element			
ifm:WBI.MEDIATION. ParameterMediation. ENTRY	ENTRY	Type	string
		TransformName	string
WBI.MEDIATION. ParameterMediation. EXIT	EXIT	Type	string
		TransformName	string
ifm:WBI.MEDIATION. ParameterMediation. FAILURE	FAILURE	Type	string
		TransformName	string
		ErrorReport	Exception

Recovery events

The event types available for the recovery component are listed.

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 event types available for the Service Component Architecture are listed.

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: Do not confuse these events with SCA-specific Application Response Measurement (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
		TARGET MODULE	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
		TARGET MODULE	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 event types available for the Selector component are listed.

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

Event Name	Event Nature	Event Contents	Type
sel:WBI.SEL. SelectionKeyExtracted	SelectionKeyExtracted	operationName	string
sel:WBI.SEL.TargetFound	TargetFound	operationName	string
		target	string



Printed in USA