WebSphere® Application Server for z/OS, Version 7.0

Monitoring

SA23-2258-00

Note

Before using this information, be sure to read the general information under “Notices” on page 273.

Compilation date: September 17, 2008

Contents

How to send your comments

Your feedback is important in helping to provide the most accurate and highest quality information.

- To send comments on articles in the WebSphere Application Server Information Center
  1. Display the article in your Web browser and scroll to the end of the article.
  2. Click on the **Feedback** link at the bottom of the article, and a separate window containing an e-mail form appears.
  3. Fill out the e-mail form as instructed, and click on **Submit feedback** .
- To send comments on PDF books, you can e-mail your comments to: **wasdoc@us.ibm.com** or fax them to 919-254-5250.

  Be sure to include the document name and number, the WebSphere Application Server version you are using, and, if applicable, the specific page, table, or figure number on which you are commenting.

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.

Changes to serve you more quickly

Print sections directly from the information center navigation

PDF books are provided as a convenience format for easy printing, reading, and offline use. The information center is the official delivery format for IBM WebSphere Application Server documentation. If you use the PDF books primarily for convenient printing, it is now easier to print various parts of the information center as needed, quickly and directly from the information center navigation tree.

To print a section of the information center navigation:

1. Hover your cursor over an entry in the information center navigation until the **Open Quick Menu** icon is displayed beside the entry.
2. Right-click the icon to display a menu for printing or searching your selected section of the navigation tree.
3. If you select **Print this topic and subtopics** from the menu, the selected section is launched in a separate browser window as one HTML file. The HTML file includes each of the topics in the section, with a table of contents at the top.
4. Print the HTML file.

For performance reasons, the number of topics you can print at one time is limited. You are notified if your selection contains too many topics. If the current limit is too restrictive, use the feedback link to suggest a preferable limit. The feedback link is available at the end of most information center pages.

Under construction!

The Information Development Team for IBM WebSphere Application Server is changing its PDF book delivery strategy to respond better to user needs. The intention is to deliver the content to you in PDF format more frequently. During a temporary transition phase, you might experience broken links. During the transition phase, expect the following link behavior:

- Links to Web addresses beginning with http:// work
- Links that refer to specific page numbers within the same PDF book work
- The remaining links will *not* work. You receive an error message when you click them

Thanks for your patience, in the short term, to facilitate the transition to more frequent PDF book updates.

Chapter 1. Overview and new features for monitoring

Use the links provided in this topic to learn about monitoring capabilities.

New for administrators: Improved monitoring and performance tuning
A section of this topic describes what is new in the area of performance tuning.

Performance: Resources for learning

Use the following links to find relevant supplemental information about performance. The information resides on IBM® and non-IBM Internet sites, whose sponsors control the technical accuracy of the information.

These links are provided for convenience. Often, the information is not specific to the IBM WebSphere® Application Server product, but is useful for understanding the product. When possible, links are provided to technical papers and Redbooks® that supplement the broad coverage of the release documentation with in-depth examinations of particular product areas. The following sections are covered in this reference:

View the following links for additional information:

- "Request metrics "
- "Monitoring performance with third-party tools"
- "Tuning performance"
- "Java™ performance resource"

Request metrics

- Systems Management: Application Response Measurement (ARM)

  The Open Group ARM specifications.

Monitoring performance with third-party tools

- IBM Search Solutions.

  Use IBM's Global Solution Directory to find a list of IBM's business partners that offer performance monitoring tools compliant with WebSphere Application Server.

Tuning performance

- Hints on Running a high-performance Web server

  Read hints about running Apache on a heavily loaded Web server. The suggestions include how to tune your kernel for the heavier TCP/IP load, and hardware and software conflicts.
- Performance Analysis for Java™ Web sites

  Offers clear explanations and expert practical guidance on performance analysis for Java-based Web sites. It offers extensive appendices, including worksheets for capacity planning, checklists to help you prepare for different stages of performance testing, and a list of performance-test tool vendors.
- WebSphere Application Server Development Best Practices for Performance and Scalability

  Describes development best practices for Web applications with servlets, JavaServer Pages files, JDBC connections, and enterprise applications with Enterprise JavaBeans™ components.

Java™ performance resource

- IBM developerWorks®

  Search the IBM developerWorks Web site for a list of garbage collection documentation, including ″Understanding the IBM Java Garbage Collector″, a three-part series. To locate the documentation, search on ″sensible garbage collection″ in the developerWorks search application.

  Review ″Understanding the IBM Java Garbage Collector″ for a description of the IBM verbose:gc output and more information about the IBM garbage collector.
- Whitepaper: IBM WebSphere Application Server for z/OS® Version 6; A performance report

The information in this white paper is designed to provide a balance of performance and benefits that can suit users of both IBM System z™ mainframes, and Java™ 2 Platform, Enterprise Edition (J2EE) technology-based IBM WebSphere® systems.

Chapter 2. Monitoring end user response time

To analyze Web site response from a client viewpoint, use Tivoli® Monitoring for Transaction Performance.

About this task

Monitoring system response time provides an external perspective of how the overall Web site responds and performs from a client viewpoint. It is important to understand the load and response time on your site. You have some options when monitoring at this level.

- Use Tivoli Monitoring for Transaction Performance to enable you to inject and monitor synthetic transactions, helping you identify when your Web site experiences a problem.
- You might also use other performance monitoring tools.

Chapter 3. Monitoring overall system health

Monitoring overall system health is fundamentally important to understanding the health of every system involved with your system. This includes Web servers, application servers, databases, back-end systems, and any other systems critical to running your Web site.

Before you begin

If any system has a problem, it might cause the `servlet is slow` message to appear. IBM and several other business partners leverage the WebSphere APIs to capture performance data and to incorporate it into an overall 24-by-7 monitoring solution. WebSphere Application Server provides Performance Monitoring Infrastructure (PMI) data to help monitor the overall health of the WebSphere Application Server environment. PMI provides average statistics on WebSphere Application Server resources, application resources, and system metrics. Many statistics are available in WebSphere Application Server, and you might want to understand the ones that most directly measure your site's resources to detect problems.

About this task

To monitor overall system health, monitor the following statistics at a minimum:

Metric	Meaning
Average response time	Include statistics, for example, servlet or enterprise beans response time. Response time statistics indicate how much time is spent in various parts of WebSphere Application Server and might quickly indicate where the problem is (for example, the servlet or the enterprise beans).
Number of requests (transactions)	Enables you to look at how much traffic is processed by WebSphere Application Server, helping you to determine the capacity that you have to manage. As the number of transactions increase, the response time of your system might be increasing, showing the need for more system resources or the need to retune your system to handle increased traffic.
Number of live HTTP sessions	The number of live HTTP sessions reflects the concurrent usage of your site. The more concurrent live sessions, the more memory is required. As the number of live sessions increase, you might adjust the session time-out values or the Java virtual machine (JVM) heap available.
Database and connection pool size	
Java virtual memory (JVM)	Use the JVM metric to understand the JVM heap dynamics, including the frequency of garbage collection. This data can assist in setting the optimal heap size. In addition, use the metric to identify potential memory leaks.
CPU	You must observe these system resources to ensure that you have enough system resources, for example, CPU, I/O, and paging, to handle the workload capacity.
I/O	
System paging	

WebSphere Application Server for z/OS relies on WLM services to collect some of the accounting and performance data.

Resource Measurement Facility (RMF™) and RMF-written System Management Facility (SMF) records present performance and accounting information to the WebSphere Application Server. In addition, the WebSphere Application Server for z/OS has SMF records that collect additional domain-specific information

Turn off the SMF records or RMF data using the administrative console and the SMFPRMxx parmlib member if you do not need the information. Use the SMFPRMxx parmlib member to control the detail of the WebSphere Application Server for z/OS SMF records. If you need SMF information, review the SMF records to ensure you are collecting only the record types and details that you need.

Setting up your workload manager goals and filtering criteria is beyond the scope of this section. You can classify work into service classes based on user ID and server name. Classify the control regions as reasonably high-performing system tasks

To monitor several of these statistics, WebSphere Application Server provides the Performance Monitoring Infrastructure to obtain the data, and provides the Tivoli Performance Viewer (TPV) in the administrative console to view this data.

1. Enable PMI through the administrative console to begin data collection.
2. Use TPV, IBM Tivoli Optimizer for WebSphere Application Server, or some other third-party performance monitoring and management solutions to monitor performance.
3. Extend monitoring capabilities by developing your own monitoring applications or extending PMI.

Performance Monitoring Infrastructure (PMI)

Use this page to learn about Performance Monitoring Infrastructure and other tools to help monitor the overall health of the application server.

A typical Web system consists of a Web server, application server, and a database. Monitoring and tuning the application server is critical to the overall performance of the Web system. Performance Monitoring Infrastructure (PMI) is the core monitoring infrastructure for WebSphere Application Server and WebSphere family products like Portal, Commerce, and so on. The performance data provided by WebSphere PMI helps to monitor and tune the application server performance.

When tuning the WebSphere Application Server for optimal performance, or fixing a poorly performing Java Platform, Enterprise Edition (Java EE) application, it is important to understand how the various run time and application resources are behaving from a performance perspective. PMI provides a comprehensive set of data that explains the runtime and application resource behavior. For example, PMI provides database connection pool size, servlet response time, Enterprise JavaBeans (EJB) method response time, Java virtual machine (JVM) garbage collection time, CPU usage, and so on. This data can be used to understand the runtime resource utilization patterns of the thread pool, connection pool, and so on, and the performance characteristics of the application components like servlets, JavaServer Pages (JSP), and enterprise beans.

Using PMI data, the performance bottlenecks in the application server can be identified and fixed. For instance, one of the PMI statistics in the Java DataBase Connectivity (JDBC) connection pool is the *number of statements discarded from prepared statement cache*. This statistic can be used to adjust the prepared statement cache size in order to minimize the discards and to improve the database query performance. PMI data can be monitored and analyzed by Tivoli Performance Viewer (TPV), other Tivoli tools, your own applications, or third party tools. TPV is a graphical viewer for PMI data that ships with WebSphere Application Server. Performance advisors use PMI data to analyze the run-time state of the application server, and provide tuning advice to optimize the application server resource utilization.

PMI data can also be used to monitor the health of the application server. Some of the health indicators are CPU usage, Servlet response time, and JDBC query time. Performance management tools like Tivoli Monitoring for Web Infrastructure and other third party tools can monitor the PMI data and generate alerts based on some predefined thresholds.

PMI architecture

The Performance Monitoring Infrastructure (PMI) uses a client-server architecture.

The server collects performance data from various WebSphere Application Server components. A client retrieves performance data from one or more servers and processes the data. WebSphere Application Server, Version 6 supports the Java Platform, Enterprise Edition (Java EE) Management Reference Implementation (JSR-77).

In WebSphere Application Server, Version 6 and later, PMI counters are enabled, based on a monitoring or instrumentation level. The levels are None, Basic, Extended, All, and Custom. These levels are specified in the PMI module XML file. Enabling the module at a given level includes all the counters at the given level plus counters from levels below the given level. So, enabling the module at the extended level enables all the counters at that level plus all the basic level counters as well.

JSR-077 defines a set of statistics for Java EE components as part of the StatisticProvider interface. The PMI monitoring level of Basic includes all of the JSR-077 specified statistics. PMI is set to monitor at a *Basic* level by default.

PMI and Java Platform, Enterprise Edition 1.4 Performance Data Framework

Java Platform, Enterprise Edition (Java EE) 1.4 includes a Performance Data Framework that is defined as part of JSR-077 (Java 2 Platform, Enterprise Edition Management Specification).

This framework specifies the performance data that must be available for various Java EE components. WebSphere PMI complies with Java EE 1.4 standards by implementing the Java EE 1.4 Performance Data Framework.

In addition to providing statistics that are defined in Java EE 1.4, PMI provides additional statistics about the Java EE components, for example, servlets and enterprise beans, and WebSphere Application Server-specific components, for example, thread pools and workload management.

PMI data classification

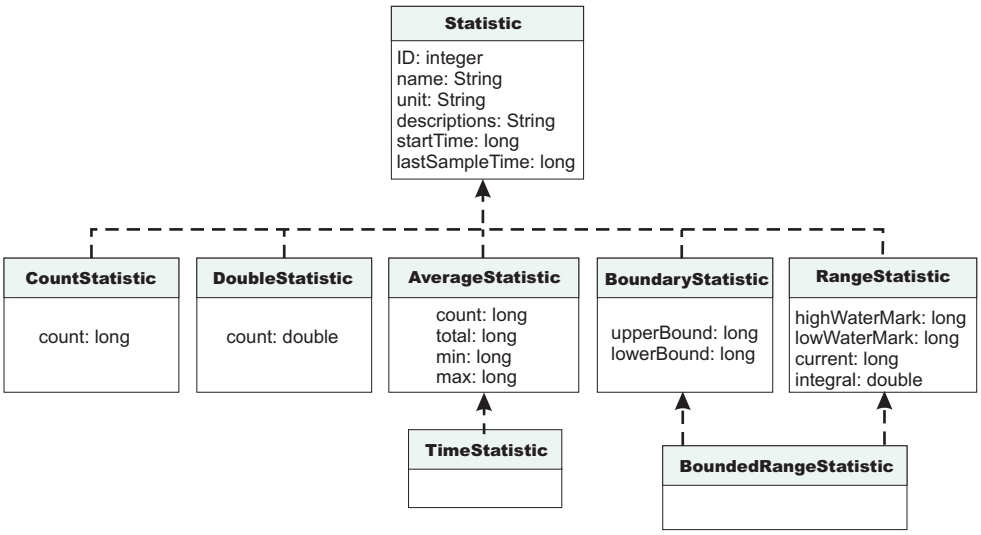
This topic describes the Performance Monitoring Infrastructure (PMI) data classification.

The static component consists of a name, ID and other descriptive attributes to identify the data. The dynamic component contains information that changes over time, such as the current value of a counter and the time stamp associated with that value.

The PMI data can be one of the following statistic types (these statistic types follow the J2EE 1.4 Performance Data Framework):

Statistic type	Description	Example
CountStatistic	Represents a running count of a given value.	Number of Servlet requests
AverageStatistic	Represents a simple average. Keeps track of total, count, min, and max. The average can be derived by total and count. (This type is WebSphere extension to J2EE Performance Data Framework)	Average HttpSession size in bytes.
TimeStatistic	Same as AverageStatistic, except that the unit of measure is milliseconds or seconds.	Average Servlet response time.
RangeStatistic	Represents a time-weighted average. Keeps track of current, low water mark, high water mark, time-weight total, and integral.	Number of concurrent Servlet requests.
BoundedRangeStatistic	Same as RangeStatistic, with lower bound and upper bound.	JDBC connection pool size.

The following diagram shows the statistic class hierarchy:



Statistic

ID A unique ID that identifies the Statistic within the given Stats (WebSphere PMI extension)

name Statistic name

unit Unit of measurement for the statistic

description
Textual description of the statistic

startTime
Time the first measurement was taken

lastSampleTime
Time the most recent measurement was taken

CountStatistic

count Count since the measurement started

DoubleStatistic

count Value since the measurement started

AverageStatistic
(WebSphere PMI extension. This is the same as the TimeStatistic defined in J2EE 1.4, except that it is used to track non-time-related measurements like byte size, etc.)

count Number of measurements

total Sum of the values of all the measurements

min Minimum value

max Maximum value

BoundaryStatistic

upperBound
Upper limit of this attribute

lowerBound
Lower limit of this attribute

RangeStatistic

current
Current® value of this attribute

lowWaterMark
Lowest value of this attribute

upperWaterMark
Highest value of this attribute

integral
Time-weighted sum of this attribute [time-weighted average = integral / (lastSampleTime - startTime] (WebSphere PMI extension)

In WebSphere Application Server, Version 4, PMI data was classified with the following types:
- **Numeric**: Maps to CountStatistic in the J2EE 1.4 specification. Holds a single numeric value that can either be a long or a double. This data type is used to keep track of simple numeric data, such as counts.
- **Stat**: Holds statistical data on a sample space, including the number of elements in the sample set, their sum, and sum of squares. You can obtain the mean, variance, and standard deviation of the mean from this data.
- **Load**: Maps to the RangeStatistic or BoundedRangeStatistic, based on J2EE 1.4 specification. This data type keeps track of a level as a function of time, including the current level, the time that level was reached, and the integral of that level over time. From this data, you can obtain the time-weighted average of that level. For example, this data type is used in the number of active threads and the number of waiters in a queue.

These PMI data types continue to be supported through the PMI client API. Statistical data types are supported through both the PMI API and Java Management Extension (JMX) API.

In WebSphere Application Server, Version 4 and Version 5, CountStatistic data require a *low* monitoring level, and TimeStatistic data require a *medium* monitoring level. RangeStatistic and BoundedRangeStatistic require a *high* monitoring level. There are a few counters that are exceptions to this rule. The average method response time, the total method calls, and active methods counters require a *high* monitoring level. The Java virtual machine counters, SerializableSessObjSize, and data tracked for each individual method (method level data) require a *maximum* monitoring level. Also, the level *maximum* enables synchronized update to all the statistic types.

WebSphere Application Server Versions 6.0 and above deprecate the monitoring levels (*Low*, *Medium*, *High*, and *Max*) and introduces fine-grained control to enable/disable statistics individually. The fine-grained control is available under the custom option. Refer to “Enabling PMI using the administrative console” on page 175 for more details.

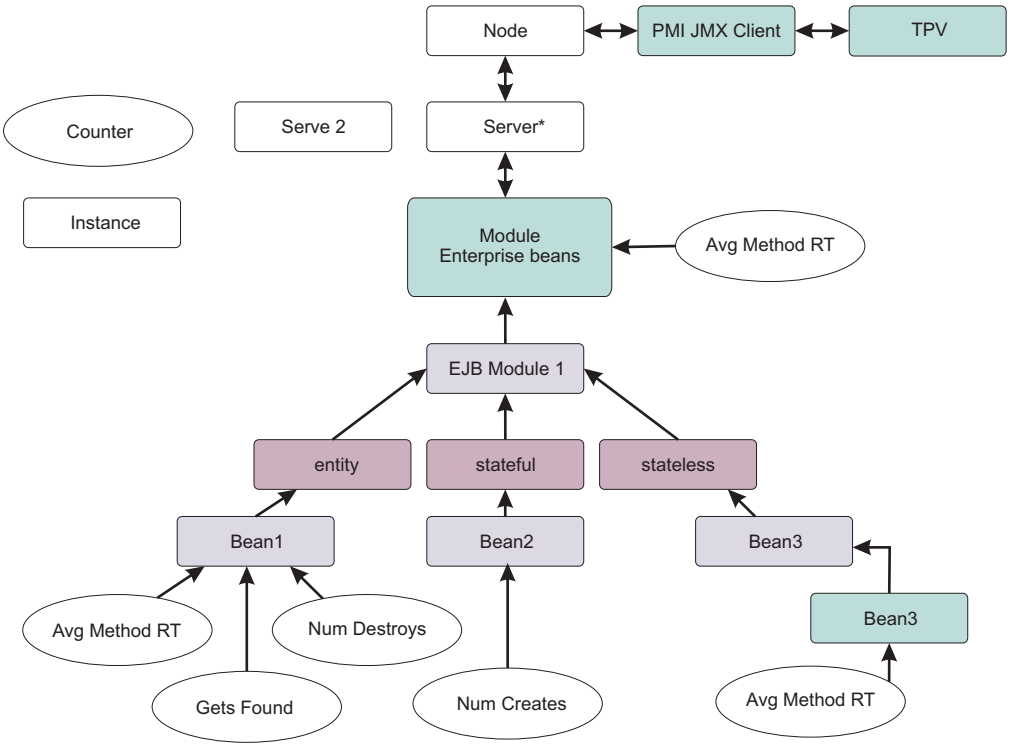
In order to reduce the monitoring overhead, updates to CountStatistic, DoubleStatistic, AverageStatistic, and TimeStatistic are not synchronized. Since this data tracks the total and average, the extra accuracy is generally not worth the performance cost. RangeStatistic and BoundedRangeStatistic are very sensitive; therefore, they are always synchronized. To enable synchronized updates for all the statistic types enable the ’Use sequential update’ option. Refer to “Enabling PMI using the administrative console” on page 175 for details.

PMI data organization

Use this page as a general overview of monitoring, data collection, and counters using Performance Monitoring Infrastructure (PMI) and Tivoli Performance Viewer (TPV).

Performance Monitoring Infrastructure (PMI) provides server-side monitoring and a client-side API to retrieve performance data. PMI maintains statistical data within the entire WebSphere Application Server domain, including multiple nodes and servers. Each node can contain one or more WebSphere Application Server. Each server organizes PMI data into modules and submodules.

Hierarchy of data collections used for performance reporting to Tivoli Performance Viewer (TPV)



Tivoli Performance Viewer, formerly Resource Analyzer, organizes performance data in a centralized hierarchy of the following objects:

- **Node.** A node represents a physical machine in the WebSphere Application Server administrative domain.
- **Server.** A server is a functional unit that provides services to clients over a network. No performance data is collected for the server itself.
- **Module.**

  A module represents one of the resource categories for which collected data is reported to the performance viewer. Each module has a configuration file in XML format. This file determines organization and lists a unique identifier for each performance data in the module. Modules include enterprise beans, JDBC connection pools, J2C connection pool, Java virtual machine (JVM) run time (including Java Virtual Machine Tool Interface (JVMTI)), servlet session manager, transaction manager, Web applications, Object Request Broker (ORB), Workload Management (WLM), and dynamic cache.
- **Submodule.** A submodule represents a fine granularity of a resource category under the module. For example, ORB thread pool is a submodule of the thread pool category. Submodules can contain other submodules.
- **Counter.** A counter is a data type used to hold performance information for analysis. Each resource category (module) has an associated set of counters. The data points within a module are queried and distinguished by the MBean ObjectNames or PerfDescriptors. Examples of counters include the number of active enterprise beans, the time spent responding to a servlet request and the number of kilobytes of available memory.

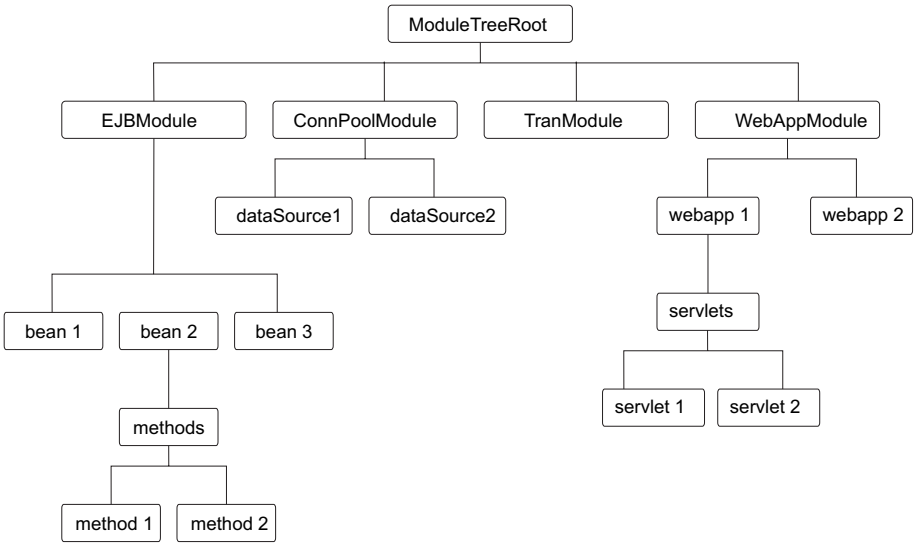
Tivoli Performance Viewer is a thin client integrated into the WebSphere Application Server administrative console. It provides a simple viewer for the performance data provided by Performance Monitoring Infrastructure (PMI), and allows users to view and manipulate the data for counters. A particular counter type can appear in several modules. For example, both the servlet and enterprise bean modules have a response time counter. In addition, a counter type can have multiple instances within a module. In the figure above, both the Enterprise beans module and Bean1 have an Avg Method RT counter.

Counters are enabled at the module level and can be enabled or disabled for elements within the module. For example, in the figure, if the enterprise beans module is enabled, its Avg Method RT counter is enabled by default. However, you can then disable the Avg Method RT counter even when the rest of the module counters are enabled. You can also, if desired, disable the Avg Method RT counter for Bean1, but the aggregate response time reported for the whole module no longer includes Bean1 data.

As part of a fine-grained control feature, WebSphere Application Server provides statistic sets which are pre-defined, fixed server-side sets, based on the PMI statistic usage scenarios. The PMI specification levels include: *none*, *basic*, *extended*, *all*, or *custom*. If you choose *none*, all PMI modules are disabled. Choosing *basic* provides the J2EE and the essential set of statistics to give you a basic level of monitoring. Selecting *extended* gives you the basic level of monitoring plus Work Load Monitor, Performance Advisor, and Tivoli resource models for a more robust monitoring set. Choosing *all* enables all statistics. Choosing *custom* gives you fine-grained control to enable or disable statistics individually.

There are only two states for a statistic: enabled or disabled. To provide an option to enable synchronized updates, WebSphere Application Server, provides a configuration parameter, synchronizedUpdate, at the PMI service level. When this attribute is true, all the statistic updates are synchronized. By default, the synchronizedUpdate parameter is set to false. You can select the **Use sequential counter updates** check box in the administrative console to enable synchronized updates. The synchronizedUpdate parameter is the equivalent to the Max level in V5.0.*x* and V5.1*x*.

Data collection can affect performance of the application server. The impact depends on the number of counters enabled, the type of counters enabled and the monitoring level set for the counters.



The following PMI modules are available to provide statistical data:

Enterprise bean module, enterprise bean, methods in a bean

Data counters for this category report load values, response times, and life cycle activities for enterprise beans. Examples include the average number of active beans and the number of times bean data is loaded or written to the database. Information is provided for enterprise bean methods and the remote interfaces used by an enterprise bean. Examples include the number of times a method is called and the average response time for the method. In addition, the Tivoli Performance Viewer reports information on the size and use of a bean objects cache or enterprise bean object pool. Examples include the number of calls attempting to retrieve an object from a pool and the number of times an object is found available in the pool.

JDBC connection pools

Data counters for this category contain usage information about the JDBC connection pools for a

database. Examples include the number of managed connections or physical connections and the total number of connections or connection handles.

Java 2 Connector (J2C) connection pool
Data counters for this category contain usage information about the Java 2 Platform, Enterprise Edition (J2EE) Connector architecture that enables enterprise beans to connect and interact with procedural back-end systems, such as Customer Information Control System (CICS®), and Information Management System (IMS™). Examples include the number of managed connections or physical connections and the total number of connections or connection handles.

Java virtual machine API (JVM)
Data counters for this category contain memory used by a process as reported by JVM run time. Examples are the total memory available and the amount of free memory for the JVM. JVM run time also includes data from the JVMTI. This data provides detailed information about the JVM running the application server.

Servlet session manager
Data counters for this category contain usage information for HTTP sessions. Examples include the total number of accessed sessions, the average amount of time it takes for a session to perform a request, and the average number of concurrently active HTTP sessions.

Thread pool
Data counters for this category contain information about the thread pools for Object Request Broker (ORB) threads and the Web container pools used to process HTTP requests. Examples include the number of threads created and destroyed, the maximum number of pooled threads allowed, and the average number of active threads in the pool.

Java Transaction API (JTA)
Data counters for this category contain performance information for the transaction manager. Examples include the average number of active transactions, the average duration of transactions, and the average number of methods per transaction.

Web applications, servlet
Data counters for this category contain information for the selected server. Examples include the number of loaded servlets, the average response time for completed requests, and the number of requests for the servlet.

Object Request Broker (ORB)
Data counters for this category contain information for the ORB. Examples include the object reference lookup time, the total number of requests, and the processing time for each interceptor.

Web services gateway (WSGW)
Data counters for this category contain information for WSGW. Examples include the number of synchronous and asynchronous requests and responses.

System data
Data counters for this category contain information for a machine (node). Examples include the CPU utilization and memory usage. Note that this category is available at the node level, which means it is only available for the node agent if you are running multiple servers.

Workload Management (WLM)
Data counters for this category contain information for workload management. Examples include the number of requests, the number of updates and average response time.

Dynamic cache
Data counters for this category contain information for the dynamic cache service. Examples include in-memory cache size, the number of invalidations, and the number of hits and misses.

Web services
Data counters for this category contain information for the Web services. Examples include the number of loaded Web services, the number of requests delivered and processed, the request response time, and the average size of requests.

Alarm manager
Data counters for this category contain information for the Alarm Manager.

Object pool
Data counters for this category contain information for Object Pools.

Scheduler
Data counters for this category contain information for the Scheduler service.

You can access PMI data through the getStatsObject and the getStatsArray method in the PerfMBean. You need to pass the MBean ObjectName(s) to the PerfMBean.

Use the following MBean types to get PMI data in the related categories:

- DynaCache: dynamic cache PMI data
- EJBModule*: Enterprise JavaBeans (EJB) module PMI data (BeanModule)
- EntityBean*: specific EJB PMI data (BeanModule)
- JDBCProvider*: JDBC connection pool PMI data
- J2CResourceAdapter*: Java 2 Connectivity (J2C) connection pool PMI data
- JVM: Java virtual machine PMI data
- MessageDrivenBean*: specific EJB PMI data (BeanModule)
- ORB: Object Request Broker PMI data
- Server: PMI data in the whole server, you must pass recursive=true to PerfMBean
- SessionManager*: HTTP Sessions PMI data
- StatefulSessionBean*: specific EJB PMI data (BeanModule)
- StatelessSessionBean*: specific EJB PMI data (BeanModule)
- SystemMetrics: system level PMI data
- ThreadPool*: thread pool PMI data
- TransactionService: JTA Transaction PMI data
- WebModule*: Web application PMI data
- Servlet*: servlet PMI data
- WLMAppServer: Workload Management PMI data
- WebServicesService: Web services PMI data
- WSGW*: Web services gateway PMI data

To use the AdminClient API to query the MBean ObjectName for each MBean type. You can either query all the MBeans and then match the MBean type or use the query String for the type only: `String query = "WebSphere:type=mytype,node=mynode,server=myserver,*";`

Set the mytype, mynode, and myserver values accordingly. You get a Set value when you call the AdminClient class to query MBean ObjectNames. This response means that you can get multiple ObjectNames.

In the previous example, the MBean types with a star (*) mean that there can be multiple ObjectNames in a server for the same MBean type. In this case, the ObjectNames can be identified by both type and name (but mbeanIdentifier is the real UID for MBeans). However, the MBean names are not predefined. They are decided at run time based on the applications and resources. When you get multiple ObjectNames, you can construct an array of ObjectNames that you are interested in. Then you can pass the ObjectNames to PerfMBean to get PMI data. You have the recursive and non-recursive options. The recursive option returns Stats and sub-stats objects in a tree structure while the non-recursive option returns a Stats object for that MBean only. More programming information can be found in “Developing your own monitoring applications” on page 187.

Enterprise bean counters

Use this page as a reference for properties of enterprise bean counters.

Counters for this category report load values, response times, and life cycle activities for enterprise beans.

Counter definitions*:*

Name	Key	ID	Description	Granularity	Type	Level	Overhead
CreateCount	beanModule.creates	1	The number of times that beans were created	Per home	CountStatistic	Basic	Low
RemoveCount	beanModule.removes	2	The number of times that beans were removed	Per home	CountStatistic	Basic	Low
ActivateCount	beanModule.activates	3	The number of times that beans were activated (entity and stateful)	Per home	CountStatistic	All	Low
PassivateCount	beanModule.passivates	4	The number of times that beans were passivated (entity and stateful)	Per home	CountStatistic	Basic	Low
InstantiateCount	beanModule.instantiates	5	The number of times that bean objects were instantiated	Per home	CountStatistic	All	Low
FreedCount	beanModule.destroys	6	The number of times that bean objects were freed	Per home	CountStatistic	All	Low
LoadCount	beanModule.loads	7	The number of times that bean data was loaded from persistent storage (entity)	Per home	CountStatistic	All	Low
StoreCount	beanModule.stores	8	The number of times that bean data was stored in persistent storage (entity)	Per home	CountStatistic	All	Low

Ready Count	beanModule.readyCount	9	The number of concurrently ready beans (entity and session). This counter was called concurrent active in Versions 3.5.5+ and 4.0.	Per home	RangeStatistic	Basic	High
LiveCount	beanModule.concurrentLives	10	The number of concurrently live beans	Per home	RangeStatistic	Extended	High
MethodCallCount	beanModule.totalMethodCalls	11	The total number of method calls	Per home	CountStatistic	Basic	High
MethodResponseTime	beanModule.avgMethodRt	12	The average response time in milliseconds on the bean methods (home, remote, local)	Per home	TimeStatistic	Basic	High
CreateTime	beanModule.avgCreateTime	14	The average time in milliseconds that a bean create call takes including the time for the load if any	Per home	TimeStatistic	All	Max
RemoveTime	beanModule.avgRemoveTime	15	The average time in milliseconds that a bean entries call takes including the time at the database, if any	Per home	TimeStatistic	All	Max
ActiveMethodCount	beanModule.activeMethods	18	The number of concurrently active methods - the number of methods called at the same time.	Per home	TimeStatistic	All	High

RetrieveFromPoolCount	beanModule.getsFromPool	19	The number of calls retrieving an object from the pool (entity and stateless)	Per home and per object pool	CountStatistic	All	Low
RetrieveFromPoolSuccessCount	beanModule.getsFound	20	The number of times that a retrieve found an object available in the pool (entity and stateless)	Per home and per object pool	CountStatistic	All	Low
ReturnsToPoolCount	beanModule.returnsToPool	21	The number of calls returning an object to the pool (entity and stateless)	Per home and per object pool	CountStatistic	Extended	Low
ReturnsDiscardCount	beanModule.returnsDiscarded	22	The number of times that the returning object was discarded because the pool was full (entity and stateless)	Per home and per object pool	CountStatistic	Extended	Low
DrainsFromPoolCount	beanModule.drainsFromPool	23	The number of times that the daemon found the pool was idle and attempted to clean it (entity and stateless)	Per home and per object pool	CountStatistic	All	Low
DrainSize	beanModule.avgDrainSize	24	The average number of objects discarded in each drain (entity and stateless)	Per home and per object pool	TimeStatistic	All	Medium
PooledCount	beanModule.poolSize	25	The number of objects in the pool (entity and stateless)	Per home and per object pool	RangeStatistic	Basic	High
MessageCount	beanModule.messageCount	26	The number of messages delivered to the bean onMessage method (message driven beans)	Per type	CountStatistic	Basic	Low

MessageBackoutCount	beanModule.messageBackoutCount	27	The number of messages that failed to be delivered to the bean onMessage method (message driven beans)	Per type	CountStatistic	All	Low
WaitTime	beanModule.avgSrvSessionWaitTime	28	The average time to obtain a ServerSession from the pool (message-driven bean)	Per type	TimeStatistic	All	Medium
ServerSessionPoolUsage	beanModule.serverSessionUsage	29	The percentage of the server session pool in use (message driven)	Per type	RangeStatistic	All	High
ActivationTime	beanModule.activationTime	30	The average time in milliseconds that a beanActivate call takes including the time at the database, if any	Per home	TimeStatistic	All	Medium
PassivationTime	beanModule.passivationTime	31	The average time in milliseconds that a beanPassivate call takes including the time at the database, if any	Per home	TimeStatistic	All	Medium
LoadTime	beanModule.loadTime	32	The average time in milliseconds for loading the bean data from persistent storage (entity)	Per home	TimeStatistic	All	Medium

StoreTime	beanModule.storeTime	33	The average time in milliseconds for storing the bean data to persistent storage (entity)	Per home	TimeStatistic	All	Medium
PassivationCount	beanModule.passivationCount	34					
ReadyCount	beanModule.methodReadyCount	35					
MethodCalls	beanModule.methods.methodCalls	51					
MethodRt	beanModule.methods.methodRt	52					
MethodLoad	beanModule.methods.methodLoad	53					
MethodLevelCallCount	beanModule.methodLevelCallCount		The number of method invocations made by WebSphere Application Server on the enterprise bean. For message driven beans, this is the number of attempts to deliver messages to the bean onMessage method.	Per type	CountStatistic	Basic	Low

JDBC connection pool counters

Performance Monitoring Infrastructure (PMI) collects performance data for 4.0 and 5.0 Java Database Connectivity (JDBC) data sources. For a 4.0 data source, the data source name is used. For a 5.0 data source, the Java Naming and Directory Interface (JNDI) name is used.

The JDBC connection pool counters are used to monitor the performance of JDBC data sources. You can find the data by using the Tivoli performance viewer and looking under each application server by click ***application_server*** > **JDBC connection pool**.

Note: With the instrumentation level set to anything other than MAX, the values may be less accurate for TimeStatistics and CountStatistics (and in the case of CountStatistics, such as numConnectionHandles, can even be negative). This is due to counters not being synchronized. Synchronizing counters is very expensive in terms of resources, so it is only done when the instrumentation level is set to MAX.

Counter definitions*:*

Name	Key	ID	Description	Granularity	Type	Level	Overhead
Num creates	connectionPoolModule.numCreates	1	The total number of connections created	Per connection pool	CountStatistic	All	Low
Num closes	connectionPoolModule.numDestroys	2	The total number of connections closed.	Per connection pool	CountStatistic	All	Low
Num allocates	connectionPoolModule.numAllocates	3	The total number of connections allocated	Per connection pool	CountStatistic	All	Low
Num returns	connectionPoolModule.numReturns	4	The total number of connections returned	Per connection pool	CountStatistic	All	Low
Pool size	connectionPoolModule.poolSize	5	The size of the connection pool	Per connection pool	BoundedRangeStatistic	All	High
Free pool size	connectionPoolModule.freePoolSize	6	The number of free connections in the pool (apply to 5.0 DataSource only)	Per connection pool	BoundedRangeStatistic	Basic	High
Concurrent waiters	connectionPoolModule.concurrentWaiters	7	The number of threads that are currently waiting for a connection	Per connection pool	RangeStatistic	All	High
Faults	connectionPoolModule.faults	8	The total number of faults, such as timeouts, in the connection pool	Per connection pool	CountStatistic	All	Low
Percent used	connectionPoolModule.percentUsed	9	The average percent of the pool that is in use	Per connection pool	RangeStatistic	Basic	High
Percent maxed	connectionPoolModule.percentMaxed	10	The average percent of the time that all connections are in use	Per connection pool	RangeStatistic	All	High
Avg use time (ms)	connectionPoolModule.avgUseTime	12	The average time a connection is used (apply to 5.0 DataSource only). Difference between the time at which the connection is allocated and returned. This value includes the JDBC operation time.	Per connection pool	TimeStatistic	All	Medium
Avg wait time (ms)	connectionPoolModule.avgWaitTime	13	The average waiting time in milliseconds until a connection is granted	Per connection pool	TimeStatistic	All	Medium
Num managed connections	connectionPoolModule.numManagedConnections	14	The number of ManagedConnection objects in use for a particular connection pool (applies to V5.0 DataSource objects only)	Per connection factory	CountStatistic	All	Low
Num connection handles	connectionPoolModule.numConnectionHandles	15	The number of Connection objects in use for a particular connection pool (apply to 5.0 DataSource only)	Per connection factory	CountStatistic	All	Low

Prepared stmt cache discards	connectionPoolModule.prepStmtCacheDiscards	21	The total number of statements discarded by the least recently used (LRU) algorithm of the statement cache	Per connection pool	CountStatistic	All	Low
JDBC time	connectionPoolModule.jdbcOperationTimer	22	The amount of time in milliseconds spent running in the JDBC driver which includes time spent in the JDBC driver, network, and database (apply to 5.0 DataSource only)	Per data source	TimeStatistic	All	Medium

J2C connection pool counters

Use this page as a reference for properties of J2C connection pool counters.

The Java 2 Connector (J2C) connection pool counters are used to monitor J2C connection pool performance. You can find the data using the Tivoli performance viewer and clicking ***application_server*** > **J2C connection pool**.

Counter definitions*:*

Name	Key	Description	Version	Granularity	Type	Level	Overhead	ID
ManagedConnectionCount	j2cModule.numManagedConnections	The number of ManagedConnection objects in use	5.0	Per connection factory	CountStatistic	All	Low	14
ConnectionHandleCount	j2cModule.numConnectionHandles	The number of connections that are associated with ManagedConnections (physical connections) objects in this pool	5.0	Per connection factory	CountStatistic	All	Low	15
CreateCount	j2cModule.numManagedConnectionsCreated	The total number of managed connections created	5.0	Per connection factory	CountStatistic	Basic	Low	1
CloseCount	j2cModule.numManagedConnectionsDestroyed	The total number of managed connections destroyed	5.0	Per connection factory	CountStatistic	Basic	Low	2
AllocateCount	j2cModule.numManagedConnectionsAllocated	The total number of times that a managed connection is allocated to a client (the total is maintained across the pool, not per connection).	5.0	Per connection factory	CountStatistic	All	Low	3
FreedCount	j2cModule.numManagedConnectionsReleased	The total number of times that a managed connection is released back to the pool (the total is maintained across the pool, not per connection).	5.0	Per connection factory	CountStatistic	All	Low	4
FaultCount	j2cModule.faults	The number of faults, such as timeouts, in the connection pool	5.0	Per connection factory	CountStatistic	All	Low	8
FreePoolSize	j2cModule.freePoolSize	The number of free connections in the pool	5.0	Per connection factory	BoundedRangeStatistic	Basic	High	6
PoolSize	j2cModule.poolSize	Average number of managed connections in the pool.	5.0	Per connection factory	BoundedRangeStatistic	Basic	High	5
WaitingThreadCount	j2cModule.concurrentWaiters	Average number of threads concurrently waiting for a connection	5.0	Per connection factory	RangeStatistic	Basic	High	7
PercentUsed	j2cModule.percentUsed	Average percent of the pool that is in use. The value is determined by the total number of configured connections in the ConnectionPool, not the current number of connections.	5.0	Per connection factory	RangeStatistic	All	High	9

PercentMaxed	j2cModule.percentMaxed	Average percent of the time that all connections are in use	5.0	Per connection factory	RangeStatistic	All	High	10
WaitTime	j2cModule.avgWait	Average waiting time in milliseconds until a connection is granted	5.0	Per connection factory	TimeStatistic	Basic	Medium	13
UseTime	j2cModule.useTime	Average time in milliseconds that connections are in use	5.0	Per connection factory	TimeStatistic	Basic	Medium	12

Java virtual machine counters

You can use the Java virtual machine (JVM) counters that the Performance Monitoring Infrastructure (PMI) and Tivoli Performance Viewer (TPV) collect to monitor JVM performance.

The total, used, and free heap size counters are available without any additional configuration settings. The remaining counters are available only when a Java virtual machine profiler is enabled.

Counter definitions:

Name	Key	Description	Granularity	Type	Level	Overhead	ID
FreeMemory	jvmRuntimeModule.freeMemory	The free memory in the JVM run time	Per JVM	CountStatistic	Extended	Low	2
HeapSize	jvmRuntimeModule.totalMemory	The total memory in the JVM run time	Per JVM	BoundedRangeStatistic. The upperBound and lowerBound are not implemented for the Total memory counter.	Basic	High	1
ProcessCpuUsage	jvmRuntimeModule.cpuUsage	The CPU Usage (in percent) of the Java virtual machine.	Per JVM	CountStatistic	Basic	Low	5
UpTime	jvmRuntimeModule.upTime	The amount of time that the JVM is running	Per JVM	CountStatistic	Basic	Low	4
UsedMemory	jvmRuntimeModule.usedMemory	The used memory in the JVM run time	Per JVM	CountStatistic	Basic	Low	3
GCcount	jvmRuntimeModule.numGcCalls	The number of garbage collection calls. This counter is not available unless the JVM profiler is enabled.	Per JVM	CountStatistic	All	Max	11
GCIntervalTime	jvmRuntimeModule.avgTimeBetweenGcCalls	The average garbage collection value in seconds between two garbage collections. This counter is not available unless the JVM profiler is enabled.	Per JVM	TimeStatistic	All	Max	12
GCTime	jvmRuntimeModule.avgGcDuration	The average duration of a garbage collection. This counter is not available unless the JVM profiler is enabled.	Per JVM	TimeStatistic	All	Max	13
WaitsForLockCount	jvmRuntimeModule.numWaitsForLock	The number of times that a thread waits for a lock. This counter is not available unless the JVM profiler is enabled.	Per JVM	CountStatistic	All	Max	19
WaitForLockTime	jvmRuntimeModule.avgTimeWaitForLock	The average time that a thread waits for a lock. This counter is not available unless the JVM profiler is enabled.	Per JVM	TimeStatistic	All	Max	20
ThreadStartedCount	jvmRuntimeModule.numThreadsStarted	The number of threads started. This counter is not available unless the JVM profiler is enabled.	Per JVM	CountStatistic	All	Max	17
ThreadEndedCount	jvmRuntimeModule.numThreadsDead	The number of failed threads. This counter is not available unless the JVM profiler is enabled.	Per JVM	CountStatistic	All	Max	18

ObjectAllocateCount	jvmRuntimeModule.numObjectsAllocated **Note:** This counter only applies to Versions 4.0 - 6.0.x. It was deprecated in Version 6.1.	The number of objects that are allocated in the heap. This counter is not available unless the -XrunpmiJvmpiProfiler option is set when starting the JVM.	Per JVM	CountStatistic	All	Max	14
ObjectMovedCount	jvmRuntimeModule.numObjectsMoved **Note:** This counter only applies to Versions 4.0 - 6.0.x. It was deprecated in Version 6.1.	The number of objects in the heap. This counter is not available unless the -XrunpmiJvmpiProfiler option is set when starting the JVM.	Per JVM	CountStatistic	All	Max	16
ObjectFreedCount	jvmRuntimeModule.numObjectsFreed **Note:** This counter only applies to Versions 4.0 - 6.0.x. It was deprecated in Version 6.1.	The number of objects freed in the heap. This counter is not available unless the -XrunpmiJvmpiProfiler option is set when starting the JVM.	Per JVM	CountStatistic	All	Max	15

Note: The statistics that are gathered through the JVM Tool Interface (JVMTI) are different for the JVM provided by IBM than the statistics that are gathered are for the Sun HotSpot-based JVM, including Sun HotSpot JVM on Solaris and the JVM provided by Hewlett-Packard for HP-UX.

Object Request Broker counters

You can use the Object Request Broker (ORB) counters that the Performance Monitoring Infrastructure (PMI) and Tivoli Performance Viewer (TPV) collect to monitor ORB operations.

Counter definitions:

Name	Key	Description	Granularity	Type	Level	Overhead	ID
ConcurrentRequestCount	orbPerfModule.concurrentRequests	The number of requests that are concurrently processed by the ORB	ORB	RangeStatistic	All	High	3
LookupTime	orbPerfModule.referenceLookupTime	The amount of time, in milliseconds, (normally < 10 milliseconds) that it takes to look up an object reference before a method dispatch can be completed. An excessively long time might indicate that there is an EJB container lookup problem.	Object Request Broker (ORB)	TimeStatistic	All	Medium	1
RequestCount	orbPerfModule.totalRequests	The total number of requests sent to the ORB	ORB	CountStatistic	All	Low	2
ProcessingTime	orbPerfModule.interceptors.processingTime	The time (in milliseconds) it takes a registered portable interceptor to run	Per interceptor	TimeStatistic	All	Medium	11

Servlet session counters

Use this page as a reference for properties of servlet session counters.

Data counters for this category contain usage information for HTTP sessions.

Counter definitions*:*

Name	Key	ID	Description	Granularity	Type	Level	Overhead
CreateCount	servletSessionsModule.createdSessions	1	The number of sessions that were created	Per Web application	CountStatistic	All	Low
InvalidateCount	servletSessionsModule.invalidatedSessions	2	The number of sessions that were invalidated	Per Web application	CountStatistic	All	Low
LifeTime	servletSessionsModule.sessionLifeTime	4	The average session life time in milliseconds (time invalidated - time created)	Per Web application	TimeStatistic	Extended	Medium
ActiveCount	servletSessionsModule.activeSessions	6	The number of concurrently active sessions. A session is active if the WebSphere Application Server is currently processing a request that uses that session.	Per Web application	RangeStatistic	All	High
LiveCount	servletSessionsModule.liveSessions	7	The number of local sessions that are currently cached in memory.	Per Web application	RangeStatistic	Basic	High
NoRoomForNewSessionCount	servletSessionsModule.noRoomForNewSession	8	Applies only to session in memory with AllowOverflow=false. The number of times that a request for a new session cannot be handled because it exceeds the maximum session count.	Per Web application	CountStatistic	Extended	Low
CacheDiscardCount	servletSessionsModule.cacheDiscards	9	The number of session objects that have been forced out of the cache. A least recently used (LRU) algorithm removes old entries to make room for new sessions and cache misses. Applicable only for persistent sessions.	Per Web application	CountStatistic	All	Low

ExternalReadTime	servletSessionsModule.externalReadTime	10	The time (milliseconds) taken in reading the session data from the persistent store. For multirow sessions, the metrics are for the attribute; for single row sessions, the metrics are for the entire session. Applicable only for persistent sessions. When using a JMS persistent store, you can choose to serialize the replicated data. If you choose not to serialize the data, the counter is not available.	Per Web application	TimeStatistic	Extended	Medium
ExternalReadSize	servletSessionsModule.externalReadSize	11	Size of the session data read from persistent store. Applicable only for (serialized) persistent sessions; similar to external Read Time.	Per Web application	TimeStatistic	Extended	Medium
ExternalWriteTime	servletSessionsModule.externalWriteTime	12	The time (milliseconds) taken to write the session data from the persistent store. Applicable only for (serialized) persistent sessions. Similar to external Read Time.	Per Web application	TimeStatistic	Extended	Medium
ExternalWriteSize	servletSessionsModule.externalWriteSize	13	The size of the session data written to persistent store. Applicable only for (serialized) persistent sessions. Similar to external Read Time.	Per Web application	TimeStatistic	Extended	Medium

AffinityBreakCount	servletSessionsModule.affinityBreaks	14	The number of requests that are received for sessions that were last accessed from another Web application. This value can indicate failover processing or a corrupt plug-in configuration.	Per Web application	CountStatistic	All	Low
TimeSinceLastActivated	servletSessionsModule.timeSinceLastActivated	15	The time difference in milliseconds between previous and current access time stamps. Does not include session time out.	Per Web application	TimeStatistic	All	Medium
TimeoutInvalidationCount	servletSessionsModule.invalidatedViaTimeout	16	The number of sessions that are invalidated by timeout.	Per Web application	CountStatistic	All	Low
ActivateNonExistSessionCount	servletSessionsModule.activateNonExistSessions	17	The number of requests for a session that no longer exists, presumably because the session timed out. Use this counter to help determine if the timeout is too short.	Per Web application	CountStatistic	All	Low
SessionObjectSize	servletSessionsModule.serializableSessObjSize	18	The size in bytes of (the serializable attributes of) in-memory sessions. Only session objects that contain at least one serializable attribute object is counted. A session can contain some attributes that are serializable and some that are not. The size in bytes is at a session level.	Per Web application	TimeStatistic	All	Max

Transaction counters

Use this page as a reference for properties of transaction counters.

Counter definitions*:*

Name	Key	ID	Description	Granularity	Type	Level	Overhead
GlobalBegunCount	transactionModule.globalTransBegun	1	The total number of global transactions started on the server	Per transaction manager or server	CountStatistic	Extended	Low
GlobalInvolvedCount	transactionModule.globalTransInvolved	2	The total number of global transactions involved on the server (for example, begun and imported)	Per transaction manager or server	CountStatistic	All	Low
LocalBegunCount	transactionModule.localTransBegun	3	The total number of local transactions started on the server	Per transaction manager or server	CountStatistic	Extended	Low
ActiveCount	transactionModule.activeGlobalTrans	4	The number of concurrently active global transactions	Per transaction manager or server	CountStatistic	Basic	Low
LocalActiveCount	transactionModule.activeLocalTrans	5	The number of concurrently active local transactions	Per transaction manager or server	CountStatistic	All	Low
GlobalTranTime	transactionModule.globalTranDuration	6	The average duration of global transactions	Per transaction manager or server	TimeStatistic	Extended	Medium
LocalTranTime	transactionModule.localTranDuration	7	The average duration of local transactions	Per transaction manager or server	TimeStatistic	Extended	Medium
GlobalBeforeCompletionTime	transactionModule.globalBeforeCompletionDuration	8	The average duration of before_completion for global transactions	per transaction manager or server	TimeStatistic	All	Medium
GlobalPrepareTime	transactionModule.globalPrepareDuration	9	The average duration of prepare for global transactions	Per transaction manager or server	TimeStatistic	All	Medium
GlobalCommitTime	transactionModule.globalCommitDuration	10	The average duration of commit for global transactions	Per transaction manager or server	TimeStatistic	All	Medium
LocalBeforeCompletionTime	transactionModule.localBeforeCompletionDuration	11	The average duration of before_completion for local transactions	Per transaction manager or server	TimeStatistic	All	Medium
LocalCommitTime	transactionModule.localCommitDuration	12	The average duration of commit for local transactions	Per transaction manager or server	TimeStatistic	All	Medium
OptimizationCount	transactionModule.numOptimization	13	The number of global transactions converted to single phase for optimization	Per transaction manager or server	CountStatistic	All	Low
CommittedCount	transactionModule.globalTransCommitted	14	The total number of global transactions committed	Per transaction manager or server	CountStatistic	Basic	Low

LocalCommittedCount	transactionModule.localTransCommitted	15	The number of local transactions committed	Per transaction manager or server	CountStatistic	All	Low
RolledbackCount	transactionModule.globalTransRolledBack	16	The total number of global transactions rolled back	Per transaction manager or server	CountStatistic	Basic	Low
LocalRolledbackCount	transactionModule.localTransRolledBack	17	The number of local transactions rolled back	Per transaction manager or server	CountStatistic	All	Low
GlobalTimeoutCount	transactionModule.globalTransTimeout	18	The number of global transactions timed out	Per transaction manager or server	CountStatistic	Extended	Low
LocalTimeoutCount	transactionModule.localTransTimeout	19	The number of local transactions timed out	Per transaction manager or server	CountStatistic	Extended	Low

Web application counters

Data counters for this category contain information for the selected server.

Counter definitions*:*
Web application counters

Name	Key	Description	Granularity	Type	Level	Overhead
LoadedServletCount	webAppModule.numLoadedServlets	The number of loaded servlets.	Per Web application	CountStatistic	All	Low
ReloadCount	webAppModule.numReloads	The number of reloaded servlets.	Per Web application	CountStatistic	All	Low

Servlet counters

Name	Key	ID	Description	Granularity	Type	Level	Overhead
RequestCount	webAppModule.servlets.totalRequests	11	The total number of requests that a servlet processed.	Per servlet	CountStatistic	Basic	Low
ConcurrentRequests	webAppModule.servlets.concurrentRequests	12	The number of requests that are concurrently processed.	Per servlet	RangeStatistic	Extended	High
ServiceTime	webAppModule.servlets.responseTime	13	The response time, in milliseconds, of a servlet request.	Per servlet	TimeStatistic	Basic	Medium
ErrorCount	webAppModule.servlets.numErrors	14	Total number of errors in a servlet or JavaServer Page (JSP).	Per servlet	CountStatistic	Extended	Low

URI counters

Name	Key	ID	Description	Granularity	Type	Level	Overhead
RequestCount	webAppModule.url.totalRequests	15	The total number of requests processed for an URI associated with a servlet.	Per URI	CountStatistic	Basic	Low
ConcurrentRequests	webAppModule.url.concurrentRequests	16	The number of requests processing concurrently for a URI associated with a servlet.	Per URI	RangeStatistic	Extended	High
ServiceTime	webAppModule.url.responseTime	17	The average service response time, in milliseconds, for an URI associated with a servlet.	Per URI	TimeStatistic	Basic	Medium

Asynchronous request dispatcher counter

Name	Key	ID	Description	Granularity	Type	Level	Overhead
ServiceTime	ard.averageResponseTime	1	The average response time, in milliseconds, in which an asynchronous request dispatcher (ARD) request is completed. This metric also processes asynchronous servlet and JavaServer Pages (JSP) requests that are cached by the dynamic cache service.	Per URI	TimeStatistic	Basic	Medium

Workload Management counters

You can use the workload management counters that the Performance Monitoring Infrastructure (PMI) and Tivoli Performance Viewer (TPV) collect to monitor how efficiently your workload is managed.

Data counters for this category contain information for workload management.

Counter definitions: **Client module**

Name	Key	Description	Granularity	Type	Level	Overhead	ID
ClientClusterUpdateCount	wlmModule.client.numClientClusterUpdates	The number of times initial or updated server cluster data is sent to a workload management enabled client from a server cluster member. Use this metric to determine how often cluster information is being propagated.	Per client	CountStatistic	All	Low	52
ClientResponseTime	wlmModule.client.clientResponseTime	The response time, in milliseconds, of IIOP requests being sent from a client. The response time indicates the amount of time that passes between when a request is sent from the client and a response is received from the server.	Per client	TimeStatistic	All	Medium	53
OutgoingIIOPRequestCount	wlmModule.client.numOfOutgoingRequests	The total number of outgoing IIOP requests sent from this client.	Per client	CountStatistic	All	Low	51

Server module

Name	Key	Description	Granularity	Type	Level	Overhead	ID
ConcurrentRequestCount	wlmModule.server.numOfConcurrentRequests	The number of remote IIOP requests currently being processed by this server	Per server	RangeStatistic	Extended	High	18
IIOPRequestCount	wlmModule.server.numIncomingRequests	The number of incoming IIOP requests that this server has received.	Per server	CountStatistic	Extended	Low	11
NoAffinityIIOPRequestCount	wlmModule.server.numIncomingNonAffinityRequests	The number of incoming IIOP requests that this server has received that have no affinity. These requests were sent to this server based on workload management selection policies that were decided in the workload management run time of the client.	Per server	CountStatistic	All	Low	14
NonWLMEnabledIIOPRequestCount	wlmModule.server.numIncomingNonWLMObjectRequests	The number of incoming IIOP requests that this server has received with no workload management enablement.	Per server	CountStatistic	All	Low	15
ServerClusterUpdateCount	wlmModule.server.numServerClusterUpdates	The number of times that this server received new server cluster information. This metric determines how often cluster information is being propagated.	Per server	CountStatistic	All	Low	16
ServerResponseTime	wlmModule.server.serverResponseTime	The amount of time, in milliseconds, that it takes for IIOP requests to be serviced by this server. The response time is the amount of time that passes between when the server receives the request and when it sends a response back to the client.	Per server	TimeStatistic	Extended	Medium	19

StrongAffinityIIOPRequestCount	wlmModule.server.numIncomingStrongAffinityRequests	The number of incoming IIOP requests that this server has received that have strong affinity. A strong affinity request is defined as a request that must be serviced by this application server because of a dependency that resides on the server. This request could not successfully be serviced on another member in the server cluster. In Version 5.0 ND edition, transactional affinity is the only example of a strong affinity.	Per server	CountStatistic	All	Low	12
WLMClientsServicedCount	wlmModule.server.numOfWLMClientsServiced	The number of workload management-enabled clients that this server has serviced.	Per server	CountStatistic	All	Low	17

System counters

You can use the system counters that the Performance Monitoring Infrastructure (PMI) and Tivoli Performance Viewer (TPV) collect to monitor how efficiently your system a specific machine (node) is running.

Counters for this category contain information for a machine (node).

Counter definitions:

Name	Key	Description	Granularity	Type	Level	Overhead	ID
CPUUsageSinceLastMeasurement	systemModule.cpuUtilization	The average system CPU utilization taken over the time interval since the last reading. Because the first call is required to perform initialization, a value such as 0, which is not valid, will be returned. All subsequent calls return the expected value. On SMP machines, the value returned is the utilization averaged over all CPUs.	Per node	CountStatistic	Basic	Low	1
CPUUsageSinceServerStarted	systemModule.avgCpuUtilization	The average percent CPU Usage that is busy after the server is started.	Per node	TimeStatistic	Extended	Medium	2
FreeMemory	systemModule.freeMemory	The amount of real free memory available on the system. Real memory that is not allocated is only a lower bound on available real memory, since many operating systems take some of the otherwise unallocated memory and use it for additional I/O buffering. The exact amount of buffer memory which can be freed up is dependent on both the platform and the application(s) running on it.	Per node	CountStatistic	All	Low	3

Dynamic cache counters

Use this page as a reference for properties of dynamic cache counters.

You can use the Performance Monitoring Infrastructure (PMI) data for dynamic cache to monitor the behavior and performance of the dynamic cache service. For information on the functions and usages of dynamic cache, refer to Task overview: Using the dynamic cache service to improve performance.

Use the dynamic cache MBean to access the related data and display it under dynamic cache in the Tivoli Performance Viewer (TPV).

Counter definitions:

Name	Key	ID value	Description	Granularity	Type	Level	Overhead
MaxInMemoryCacheEntryCount	cacheModule..maxInMemoryCacheEntryCount	1	The maximum number of in-memory cache entries. Locate it under servlet instance or object instance.	Per Server	CountStatistic	All	Low
InMemoryCacheEntryCount	cacheModule.inMemoryCacheEntryCount	2	The current number of in-memory cache entries. Locate it under servlet instance and object instance.	Per Server	CountStatistic	All	Low
HitsInMemoryCount	cacheModule.hitsInMemoryCount	21	The number of requests for cacheable objects that are served from memory. For servlet instance, locate it under template group. For object instance, locate it under object group.	Per Server	CountStatistic	All	Low
HitsOnDiskCount	cacheModule.hitsOnDiskCount	22	The number of requests for cacheable objects that are served from disk. For servlet instance, locate it under template group. For object instance, locate it under object group.	Per Server	CountStatistic	All	Low
ExplicitInvalidationCount	cacheModule.explicitInvalidationCount	23	The number of explicit invalidations. For servlet instance, locate it under template group. For object instance, locate it under object group.	Per Server	CountStatistic	All	Low
LruInvalidationCount	cacheModule.lruInvalidationCount	24	The number of cache entries that are removed from memory by a Least Recently Used (LRU) algorithm. For servlet instance, locate it under template group. For object instance, locate it under object group.	Per Server	CountStatistic	All	Low

TimeoutInvalidationCount	cacheModule.timeoutInvalidationCount	25	The number of cache entries that are removed from memory and disk because their timeout has expired. For servlet instance, locate it under template group. For object instance, locate it under object group.	Per Server	CountStatistic	All	Low
InMemoryAndDiskCacheEntryCount	cacheModule.inMemoryAndDiskCacheEntryCount	26	The current number of used cache entries in memory and disk. For servlet instance, locate it under template group. For object instance, locate it under object group.	Per Server	CountStatistic	All	Low
RemoteHitCount	cacheModule.remoteHitCount	27	The number of requests for cacheable objects that are served from other Java virtual machines within the replication domain. For servlet instance, locate it under template group. For object instance, locate it under object group.	Per Server	CountStatistic	All	Low
MissCount	cacheModule.missCount	28	The number of requests for cacheable objects that were not found in the cache. For servlet instance, locate it under template group. For object instance, locate it under object group.	Per Server	CountStatistic	All	Low
ClientRequestCount	cacheModule.clientRequestCount	29	The number of requests for cacheable objects that are generated by applications running on this application server. For servlet instance, locate it under template group. For object instance, locate it under object group.	Per Server	CountStatistic	All	Low

DistributedRequestCount	cacheModule.distributedRequestCount	30	The number of requests for cacheable objects that are generated by cooperating caches in this replication domain. For servlet instance, locate it under template group. For object instance, locate it under object group.	Per Server	CountStatistic	All	Low
ExplicitMemoryInvalidationCount	cacheModule.explicitMemoryInvalidationCount	31	The number of explicit invalidations that result in the removal of an entry from memory. For servlet instance, locate it under template group. For object instance, locate it under object group.	Per Server	CountStatistic	All	Low
ExplicitDiskInvalidationCount	cacheModule.explicitDiskInvalidationCount	32	The number of explicit invalidations that result in the removal of an entry from disk. For servlet instance, locate it under template group. For object instance, locate it under object group.	Per Server	CountStatistic	All	Low
LocalExplicitInvalidationCount	cacheModule.localExplicitInvalidationCount	34	The number of explicit invalidations that are generated locally, either programmatically or by a cache policy. For servlet instance, locate it under template group. For object instance, locate it under object group.	Per Server	CountStatistic	All	Low
RemoteExplicitInvalidationCount	cacheModule.remoteExplicitInvalidationCount	35	The number of explicit invalidations that are received from a cooperating Java virtual machine in this replication domain. For servlet instance, locate it under template group. For object instance, locate it under object group.	Per Server	CountStatistic	All	Low

RemoteCreationCount	cacheModule.remoteCreationCount	36	The number of cache entries that are received from cooperating dynamic caches. For servlet instance, locate it under template group. For object instance, locate it under object group.	Per Server	CountStatistic	All	Low
ObjectsOnDisk	cacheModule.objectsOnDisk	4	The current number of cache entries on disk. locate it under disk group.	Per Server	CountStatistic	All	Low
HitsOnDisk	cacheModule.hitsOnDisk	5	The number of requests for cacheable objects that are served from disk. Locate it under disk group.	Per Server	CountStatistic	All	Low
ExplicitInvalidationsFromDisk	cacheModule.explicitInvalidationFromDisk	6	The number of explicit invalidations that result in the removal of entries from disk. Locate it under disk group.	Per Server	CountStatistic	All	Low
TimeoutInvalidationsFromDisk	cacheModule.timeoutInvalidationsFromDisk	7	The number of disk timeouts. Locate it under disk group.	Per Server	CountStatistic	All	Low
PendingRemovalFromDisk	cacheModule.pendingRemovalFromDisk	8	The current number of pending entries that are to be removed from disk. Locate it under disk group.	Per Server	CountStatistic	All	Low
DependencyIDsOnDisk	cacheModule.dependencyIdsOnDisk	9	The current number of dependency ID that are on disk. Locate it under disk group.	Per Server	CountStatistic	All	Low
DependencyIDsBufferedForDisk	cacheModule.dependencyIdsBufferedForDisk	10	The current number of dependency IDs that are buffered for the disk. Locate it under disk group.	Per Server	CountStatistic	All	Low
DependencyIDsOffloadedToDisk	cacheModule.dependencyIdsOffloadedToDisk	11	The number of dependency IDs that are offloaded to disk. It is located under disk group.	Per Server	CountStatistic	All	Low

DependencyIDBasedInvalidationsFromDisk	cacheModule.dependencyIdBasedInvalidationsFromDisk	12	The number of dependency ID-based invalidations. Locate it under disk group.	Per Server	CountStatistic	All	Low
TemplatesOnDisk	cacheModule.templatesOnDisk	13	The current number of templates that are on disk. Locate it under disk group.	Per Server	CountStatistic	All	Low
TemplatesBufferedForDisk	cacheModule.templatesBufferedForDisk	14	The current number of templates that are buffered for the disk. Locate it under disk group.	Per Server	CountStatistic	All	Low
TemplatesOffloadedToDisk	cacheModule.templatesOffloadedToDisk	15	The number of templates that are offloaded to disk. Locate it under disk group.	Per Server	CountStatistic	All	Low
TemplateBasedInvalidationsFromDisk	cacheModule.templateBasedInvalidationsFromDisk	16	The number of template-based invalidations. Locate it under disk group.	Per Server	CountStatistic	All	Low
GarbageCollectorInvalidationsFromDisk	cacheModule.garbageCollectorInvalidationsFromDisk	17	The number of garbage collector invalidations that result in the removal of entries from disk cache due to high threshold has been reached. Locate it under disk group.	Per Server	CountStatistic	All	Low
OverflowInvalidationsFromDisk	cacheModule.overflowInvalidationsFromDisk	18	The number of invalidations that result in the removal of entries from disk due to exceeding the disk cache size or disk cache size in GB limit. Locate it under disk group.	Per Server	CountStatistic	All	Low

MBean cache statistics

Use this page as a reference for properties of MBean cache statistics.

You can use the Performance Monitoring Infrastructure (PMI) data for dynamic cache to monitor the behavior and performance of the dynamic cache service. For information on the functions and usages of dynamic cache, refer to Task overview: Using the dynamic cache service to improve performance.

MBean cache statistics*:*

MBean attributes	Description
CacheHits	The total number of cache hits.
CacheLruRemoves	The number of memory-based least recently used (LRU) evictions. These correspond to the number of objects that are evicted from the memory cache, based on the LRU policy.
CacheMisses	The total number of cache misses.
CacheRemoves	The total number of cache removes.
DependencyIdsOnDisk	Metric that captures the total number of dependency identifiers that are currently on disk.
DependencyIdsBufferedForDisk	Metric that captures the number of dependency identifiers that are buffered in memory for the disk.
DependencyIdsOffloadedToDisk	Metric that captures the number of dependency IDs that are off-loaded to disk since the server startup. This is a cumulative count.
DependencyIdBasedInvalidationsFromDisk	Metric that captures the number of dependency ID-based invalidations that have occurred since server startup.
DiskCacheSizeInMB	The total size of data objects in MB.
ExplicitInvalidationsFromMemory	Metric that captures the number of explicit invalidations that result in an entry being removed from memory.
ExplicitInvalidationsFromDisk	Metric that captures the number of explicit invalidations that result in an entry being removed from disk.
ExplicitInvalidationsLocal	Metric that captures the number of explicit invalidations that are generated locally.
ExplicitInvalidationRemote	Metric that captures the number of explicit invalidations that generate from a cooperating Java virtual machine (JVM) in the cluster.
GarbageCollectorInvalidationsFromDisk	This counter increments when the garbage collector evicts entries from the disk. If an entry is not expired, this counter increments. If an entry is expired, the TotalTimeoutInvalidationsFromDisk increments.
MemoryCacheEntries	The number of cache entries in memory.
ObjectsDeleteFromDisk	Metric that captures the number of cached objects that are deleted from disk.
ObjectsDeleteFromDisk4K	Metric that captures the number of objects that have been deleted from disk and are smaller than 4K.
ObjectsDeleteFromDisk40K	Metric that captures the number of objects that have been deleted from disk and are between 4K and 40K.
ObjectsDeleteFromDisk400K	Metric that captures the number of objects that have been deleted from disk and are between 40K and 400K.
ObjectsDeleteFromDisk4000K	Metric that captures the number of objects that have been deleted from disk and are between 400K and 400K.
ObjectsDeleteFromDiskSize	Metric that captures the cumulative size of cached objects that are deleted from disk.
ObjectsOnDisk	Metric that captures the number of cache entries that are currently on disk.
ObjectsReadFromDisk	Metric that captures the number of cached objects that are read from disk.
ObjectsReadFromDisk4K	Metric that captures the number of objects that have been read from disk and are smaller than 4K.
ObjectsReadFromDisk40K	Metric that captures the number of objects that have been read from disk and are between 4K and 40K.
ObjectsReadFromDisk400K	Metric that captures the number of objects that have been read from disk and are between 40K and 400K.
ObjectsReadFromDiskSize	Metric that captures the cumulative size of cached objects that are read from disk.
ObjectsWriteToDisk	Metric that captures the number of cached objects that are written to disk.
ObjectsWriteToDisk4K	Metric that captures the number of objects that are written to disk and are smaller than 4K.
ObjectsWriteToDisk40K	Metric that captures the number of objects that are written to disk and are between 4K and 40K.
ObjectsWriteToDisk400K	Metric that captures the number of objects that are written to disk and are between 40K and 400K.
ObjectsWriteToDisk4000K	Metric that captures the number of objects that are written to disk and are between 400K and 4000K.
ObjectsWrittenToDiskSize	Metric that captures the cumulative size of cached objects that are written to disk.

OverflowInvalidationsFromDisk	This counter increments when the limit of disk cache size or disk cache size in GB is reached. It may be removed from memory and not offloaded to disk. In addition, when performing write entry to disk, write dependency ID to disk, write template to disk, if the total disk cache files in GB is over disk cache size in GB, the entry will not write to disk. This counter increments.
PendingRemovalFromDisk	Metric that captures the number of objects that have been invalidated due to an explicit event, such as an invalidation, or some implicit event, such as a timeout or eviction policy, but are yet to be removed from disk.
PushPullTableSize	Metric that captures the size of the current PushPullTable.
RemoteInvalidationNotifications	The number of remote invalidation notifications from other servers since server startup.
RemoteObjectFetchSize	The size of requests for cacheable objects that are served from other JVMs in the cluster. (PushPull mode)
RemoteObjectHits	The number of requests for cacheable objects that are served from other JVMs in the cluster. (PushPull mode)
RemoteObjectMisses	The number of requests for cacheable objects that were not found from other JVMs in the cluster. (PushPull mode)
RemoteObjectUpdates	The number of cacheable objects that are received from other JVMs in the cluster. (Push mode)
RemoteObjectUpdateSize	The size of cacheable objects that are received from other JVMs in the cluster. (Push mode)
RemoteUpdateNotifications	The number of notifications that identify that an object is updated at another server. (The number of entries in the push-pull table).
TemplateBasedInvalidationsFromDisk	Number of template-based invalidations since the server startup.
TemplatesBufferedForDisk	Number of templates that are buffered for the disk.
TimeoutInvalidationFromDisk	Metric that captures the number of timeout invalidations that result in an entry being removed from disk.
TimeoutInvalidationsFromMemory	Metric that captures the number of timeout invalidations that result in an entry being removed from memory.
TemplatesOffloadedToDisk	Number of templates that are off-loaded to disk since the server startup.
TemplatesOnDisk	Metric that captures the number of templates on disk.

Web services counters

Use this page as a reference for properties of Web services counters.

You can use the Web services counters that the Performance Monitoring Infrastructure (PMI) and Tivoli Performance Viewer (TPV) collect to monitor the performance of your Web services.

Counter definitions*:*

Name	Key	ID	Description	Granularity	Type	Level	Overhead
DispatchedRequestCount	webServicesModule.services.numberDispatched	12	The number of requests the service dispatched or delivered	Per Web service	CountStatistic	All	Low
DispatchResponseTime	webServicesModule.services.dispatchResponseTime	16	The average response time, in milliseconds, to dispatch a request	Per Web service	TimeStatistic	All	Medium
LoadedWebServiceCount	webServicesModule.numLoadedServices	1	The number of loaded Web services	Per Web service	CountStatistic	All	Low
PayloadSize	webServicesModule.services.size	18	The average payload size in bytes of a received request or reply	Per Web service	CountStatistic	All	Medium
ProcessedRequestCount	webServicesModule.services.numberSuccessfull	13	The number of requests the service successfully processed.	Per Web service	CountStatistic	All	Low
ReceivedRequestCount	webServicesModule.services.numberReceived	11	The number of requests the service received	Per Web service	CountStatistic	All	Low
ReplyPayloadSize	webServicesModule.services.replySize	20	The average payload size in bytes of a reply.	Per Web service	CountStatistic	All	Medium
ReplyResponseTime	webServicesModule.services.replyResponseTime	17	The average response time, in milliseconds, to prepare a reply after dispatch	Per Web service	TimeStatistic	All	Medium
RequestFailedEndpoint	webServiceEndpoint.numberFailed	32	The number of requests that have failed processing for the endpoint	Per Web service endpoint	CountStatistic	All	Low
RequestPayloadSize	webServicesModule.services.requestSize	19	The average payload size in bytes of a request	Per Web service	CountStatistic	All	Medium
RequestReceivedEndpoint	webServiceEndpoint.numberReceived	30	The number of requests received for the endpoint	Per Web service endpoint	CountStatistic	All	Low
RequestResponseTime	webServicesModule.services.requestResponseTime	15	The average response time, in milliseconds, to prepare a request for dispatch	Per Web service	TimeStatistic	All	Medium
RequestSizeEndpoint	webServiceEndpoint.requestSize	36	The average size, in bytes, of the request payload for the endpoint	Per Web service endpoint	AverageStatistic	All	High
RequestSizeLastEndpoint	webServiceEndpoint.requestSizeLast	38	The last size, in bytes, recorded of the request payload for the endpoint	Per Web service endpoint	CountStatistic	All	High
RequestSizeMaxEndpoint	webServiceEndpoint.requestSizeMax	37	The maximum size, in bytes, recorded of the request payload for the endpoint	Per Web service endpoint	AverageStatistic	All	High

RequestSuccessfulEndpoint	webServiceEndpoint.numberSuccessful	31	The number of requests that have successfully been processed for the endpoint	Per Web service endpoint	CountStatistic	All	Low
ResponseSizeEndpoint	webServiceEndpoint.responseSize	39	The average size, in bytes, of the response payload for the endpoint	Per Web service endpoint	AverageStatistic	All	High
ResponseSizeLastEndpoint	webServiceEndpoint.responseSizeLast	41	The last size, in bytes, recorded of the response the payload for the endpoint	Per Web service endpoint	CountStatistic	All	High
ResponseSizeMaxEndpoint	webServiceEndpoint.responseSizeMax	40	The maximum size, in bytes, recorded of the response payload for the endpoint	Per Web service endpoint	AverageStatistic	All	High
ResponseTime	webServicesModule.services.responseTime	14	The average response time, in milliseconds, for a successful request	Per Web service	TimeStatistic	All	High
ResponseTimeEndpoint	webServiceEndpoint.responseTime	33	The average amount of time, in milliseconds, between the request arrival and the response reply back to the client for the endpoint	Per Web service endpoint	TimeStatistic	All	Medium
ResponseTimeLastEndpoint	webServiceEndpoint.responseTimeLast	35	The last value, in milliseconds, recorded between the request arrival and the response reply back to the client for the endpoint	Per Web service endpoint	TimeStatistic	All	Medium
ResponseTimeMaxEndpoint	webServiceEndpoint.responseTimeMax	34	The maximum value, in milliseconds, recorded between the request arrival and the response reply back to the client for the endpoint	Per Web service endpoint	TimeStatistic	All	Medium

Alarm Manager counters

Use this page as a reference for properties of alarm manager counters.

Counters for this category contain information for the Alarm Manager.

Counter definitions*:*

Name	Key	ID values	Description	Granularity	Type	Level	Overhead
AlarmsCreatedCount	alarmManagerModule.numCreates.name	1	The total number of alarms created by all asynchronous scopes for this WorkManager	Per WorkManager	CountStatistic	All	High
AlarmsCancelledCount	alarmManagerModule.numCancelled.name	2	The number of alarms cancelled by the application	Per WorkManager	CountStatistic	All	High
AlarmsFiredCount	alarmManagerModule.numFired.name	3	The number of alarms fired	Per WorkManager	CountStatistic	All	High
AlarmLatencyDuration	alarmManagerModule.latency.name	4	The latency of alarms fired in milliseconds	Per WorkManager	RangeStatistic	All	High
AlarmsPendingSize	alarmManagerModule.alarmsPending.name	5	The number of alarms waiting to fire	Per WorkManager	RangeStatistic	All	High
AlarmRate	alarmManagerModule.alarmsPerSecond.name	6	The number of alarms firing per second	Per WorkManager	RangeStatistic	All	High

Object Pool counters

Use this page as a reference for properties of Object Pool counters.

Counters for this category contain information for Object Pools.

Counter definitions*:*

Name	Key	ID values	Description	Granularity	Type	Level	Overhead
ObjectsCreatedCount	objectPoolModule.numCreates.name	1	The total number of objects created	Per ObjectPool	CountStatistic	All	High
ObjectsAllocatedCount	objectPoolModule.numAllocates.name	2	The number of objects requested from the pool	Per ObjectPool	CountStatistic	All	High
ObjectsReturnedCount	objectPoolModule.numReturns.name	3	The number of objects returned to the pool	Per ObjectPool	CountStatistic	All	High
IdleObjectsSize	objectPoolModule.poolSize.name	4	The average number of idle object instances in the pool	Per ObjectPool	RangeStatistic	All	High

Scheduler counters

Use this page as a reference for properties of Scheduler counters.

Counters for this category contain information for the Scheduler service.

Counter definitions*:*

Name	Key	ID value	Description	Granularity	Type	Level	Overhead
TaskFailureCount	schedulerModule.failedTasks.name	1	The number of tasks that failed to run	Per Scheduler	CountStatistic	All	High
TaskFinishCount	schedulerModule.executedTasks.name	2	The number of tasks ran successfully	Per Scheduler	CountStatistic	All	High
PollCount	schedulerModule.totalPolls.name	11	The number of poll cycles completed for all daemon threads	Per Scheduler	CountStatistic	All	High
TaskFinishRate	schedulerModule.tasksPerSec.name	3	The number of tasks run per second	Per Scheduler	RangeStatistic	All	High
TaskCollisionRate	schedulerModule.collisionsPerSec.name	4	The number of collisions encountered per second between competing poll daemons	Per Scheduler	RangeStatistic	All	High
PollQueryDuration	schedulerModule.queryTime.name	5	The start time in milliseconds for each poll daemon thread's database poll query	Per Scheduler	RangeStatistic	All	High
RunDuration	schedulerModule.execTime.name	6	The time in milliseconds taken to run a task.	Per Scheduler	RangeStatistic	All	High
TaskExpirationRate	schedulerModule.taskLoadPerPoll.name	7	The number of tasks in a poll query	Per Scheduler	RangeStatistic	All	High
TaskDelayDuration	schedulerModule.execLatency.name	8	The period of time in seconds that the task is delayed	Per Scheduler	RangeStatistic	All	High
PollDuration	schedulerModule.pollTime.name	9	The number of seconds between poll cycles	Per Scheduler	RangeStatistic	All	High
TaskRunRate	schedulerModule.taskExecPerPoll.name	10	The number of tasks run by each poll daemon thread. (Multiply this by the number of poll daemon threads to get the tasks run per effective poll cycle.)	Per Scheduler	RangeStatistic	All	High

High availability manager counters

You can use the high availability manager counters that the Performance Monitoring Infrastructure (PMI) and Tivoli Performance Viewer (TPV) collect to monitor activity within your high availability environment.

Counter definitions:

Name	Key	Description	Granularity	Type	Level	Overhead	ID
BulletinBoardRebuildTime	hamanagermodule.bbMgrRebuildTime	The amount of time, in milliseconds, that is consumed rebuilding the global state of the bulletin-board. During this time, subscribers do not receive any messages. If this amount of time is too high, and is unacceptable for the desired availability, increase the number of coordinators. For proper operation of this counter, you must host the active coordinator in an application server other than the deployment manager.	Per server	TimeStatistic	All	High	1
BulletinBoardSubcriptionCount	hamanagermodule.bbMgrNumSubscriptions	The total number of bulletin board subscriptions that are managed by this server	Per server	BoundedRangeStatistic	All	High	2
BulletinBoardSubjectCount	hamanagermodule.bbMgrNumSubjects	The total number of subjects that are managed by this server	Per server	BoundedRangeStatistic	All	High	3
GroupStateRebuildTime	hamanagermodule.rebuildTime	The amount of time, in milliseconds, consumed rebuilding the global group state. During the rebuild time, no failover can happen. If this time is too high, and is unacceptable for the desired availability, increase the number of coordinators. For proper operation of this counter, you must host the active coordinator in an application server other than the deployment manager.	Per server	TimeStatistic	All	High	4
LocalBulletinBoardSubcriptionCount	hamanagermodule.bbLocalNumSubscriptions	The total number of local subject subscriptions The number includes any proxy subscriptions that the core group bridge service provides on behalf of the servers that belong to different cells.	Per server	BoundedRangeStatistic	All	High	5
LocalBulletinBoardSubjectCount	hamanagermodule.bbLocalNumSubjects	The total number of subjects that are being posted to locally. The number includes any proxy postings that the core group bridge service does on behalf of servers that belong to different cells.	Per server	BoundedRangeStatistic	All	High	6
LocalGroupCount	hamanagermodule.numLocalGroups	The total number of local groups that are in this server	Per server	Bounded RangeStatistic	All	High	7

Distribution and consistency services (DCS) stack counters

You can use the Distribution and Consistency Services (DCS) counters that the Performance Monitoring Infrastructure (PMI) and Tivoli Performance Viewer (TPV) collect to monitor communication between core groups.

Counter definitions:

Name	Key	Description	Granularity	Type	Level	Overhead	ID
CoalesceTime	DCSStats.coalesceTime	Amount of time that it actually takes to coalesce a view	Per DCS stack	TimeStatistic	All	Medium	1
HighSeverityCongestionEventCount	DCSStats.transmitterCongestedCounter	Number of times that a high severity congestion event for outgoing messages was raised.	Per DCS stack	CountStatistic	All	Medium	2
IncomingMessageSize	DCSStats.incomingMessageSize	Minimal, maximal and average size (in bytes) of the messages that were received by the DCS stack	Per DCS stack	AverageStatistic	All	High	3
JoinViewChangeTime	DCSStats.mergeTime	Amount of time that it took to do a merge view change. The DCS stack is blocked during this time.	Per DCS stack	TimeStatistic	All	High	4
LocalMemberMessageRetransmissionCount	DCSStats.numOfVSCompletionMessages	Number of messages that were retransmitted during the view change to ensure synchronization with other members.	Per DCS stack	AverageStatistic	All	High	5
MessageBufferReallocationCount	DCSStats.numOfReallocs	Number of message buffer reallocations due to inadequate buffer size. If this number is larger than 20 percent of the number of sent messages, you may want to contact IBM Support	Per DCS stack	CountStatistic	All	Medium	6
OutgoingMessageSize	DCSStats.outgoingMessageSize	Minimal, maximal, and average size (in bytes) of the messages that were sent through the DCS stack	Per DCS stack	AverageStatistic	All	High	7
ReceivedMessageCount	DCSStats.incomingMessageCounter	Number of messages received by the DCS stack	Per DCS stack	CountStatistic	All	High	8
RemoveViewChangeTime	DCSStats.splitTime	Amount of time that is consumed when splitting a group. The DCS stack is blocked during this time.	Per DCS stack	TimeStatistic	All	High	9
SentMessageCount	DCSStats.outgoingMessageCounter	Number of messages that were sent through the DCS stack	Per DCS stack	CountStatistic	All	High	10
SuspicionCount	DCSStats.suspectCounter	Number of times that the local member is suspected by other members	Per DCS stack	CountStatistic	All	High	11
SynchronizationCompleteTime	DCSStats.vsCompleteCurrentTime	Amount of time that is needed to guarantee that all view members are synchronized.	Per DCS stack	TimeStatistic	All	High	12

SynchronizationTimeoutCount	DCSStats.vsTimetoutExpiredCounter	Number of times that the synchronization procedure timed out.	Per DCS stack	CountStatistic	All	Medium	13
ViewChangeCount	DCSStats.viewCounter	Number of times that this member underwent view changes.	Per DCS stack	CountStatistic	All	Medium	14
ViewChangeTimeoutCount	DCSStats.mbrAlarmTimeoutCounter	Number of times that the view change procedure timed out.	Per DCS stack	CountStatistic	All	Medium	15
ViewGroupSize	DCSStats.groupSize	The size of the group to which the local member belongs.	Per DCS stack	AverageStatistic	All1	Medium	16

PortletContainer PMI counters

Use this page as a reference for properties of PortletContainer PMI counters.

Counter definitions

These counters are the PMI measurement points for a portlet and a portlet application.

Name	Key	ID	Description	Unit	Type	Level	Overhead
totalRequests	portletModule.totalRequests	115	The number of requests made to the portlet module.	unit.none	CountStatistic	All	Low
numErrors	portletModule.numErrors	119	The number of errors reported from the portlet module.	unit.none	CountStatistic	All	Low
concurrentRequests	portletModule.concurrentRequests	116	The number of concurrent requests.	unit.none	CountStatistic	All	High
renderResponseTime	portletModeule.renderResponseTime	117	The render response time.	unit.ms	CountStatistic	All	Medium
actionResponseTime	portletModule.actionResponseTime	118	The action response time.	unit.ms	CountStatistic	All	Medium
numLoadedPortlets	portletAppModule.numLoadedPortlets		The number of portlets loaded.	unit.none	CountStatistic	All	Low
resourceResponseTime	portletModule.resourceResponseTime	121	Response time of a JSR 286 portlet serveResource request.	unit.ms	CountStatistic	All	Medium
eventResponseTime	portletModule.eventResponseTime	120	Response time of a JSR 286 portlet processEvent request.	unit.ms	CountStatistic	All	Medium

Related tasks

Task overview: Managing portlets
You can use this task to manage deployed portlet applications.

Related reference

“PMI data organization” on page 9
Use this page as a general overview of monitoring, data collection, and counters using Performance Monitoring Infrastructure (PMI) and Tivoli Performance Viewer (TPV).

Extension registry counters

Use this page as a reference for properties of extension registry counters.

Data counters for this category contain usage information for the extension registry.

Counter definitions*:*

Name	Key	ID	Description	Granularity	Type	Level	Overhead
RequestCount	ExtensionRegistryStats.numRequests	1	The total number of cache read requests served	Per server	CountStatistic	Basic	Low
HitCount	ExtensionRegistryStats.numHits	2	The number of cache hits out of the total requests	Per server	CountStatistic	Basic	Low
HitRate	ExtensionRegistryStats.hitRate	3	The rate for cache hits	Per server	TimeStatistic	Basic	Low
DisplacementCount	ExtensionRegistryStats.numDisplaces	4	The number of cache displacements	Per server	CountStatistic	Basic	Low

Proxy counters

Use this page as a reference for properties of proxy counters.

You can use the Performance Monitoring Infrastructure (PMI) data to monitor the behavior and performance of the proxy service.

Counter definitions*:*

Name	Key	Description	Granularity	Type	Level	Overhead	ID
OutboundConnection Count	OutboundConnection Count	The total number of outbound connections since server start.	Per proxy server	CountStatistic	Basic	Low	2
ActiveOutbound ConnectionCount	ActiveOutbound ConnectionCount	The total concurrent outbound connection count at a specific time	Per proxy server	CountStatistic	Basic	Low	4
RequestCount	RequestCount	The total number of requests processed by the proxy server since server start.	Per proxy server	CountStatistic	Basic	Low	10
FailedRequestCount	FailedRequestCount	The number of failed requests due to an internal error.	Per proxy server	CountStatistic	Extended	Low	11
ProxiedRequestCount	ProxiedRequestCount	The number of requests forwarded to back-end application servers or Web servers.	Per proxy server	CountStatistic	Extended	Low	12
LocalRequestCount	LocalRequestCount	The number of requests served locally from the proxy. Includes cache hits and failed requests due to internal error.	Per proxy server	CountStatistic	Extended	Low	13
ResponseTime(TTLB)	ResponseTime(TTLB)	Proxy Server response time. The number of milliseconds that have passed before the last byte of the response was sent back to client.	Per request	TimeStatistic	Extended	Medium	20
ServerResponseTime	ServerResponseTime	Target server response time, The number of milliseconds that have passed before proxy received the first byte from target servers.	Per request	CountStatistic	Extended	Medium	21
CacheHitCount Validated	CacheHitCount Validated	The total cache hit count revalidated with target servers.	Per proxy server	CountStatistic	Basic	Low	31
CacheHitCount Unvalidated	CacheHitCount Unvalidated	The total cache hit count processed locally.	Per proxy server	CountStatistic	Basic	Low	32
CacheMissCount	CacheMissCount	The total cache miss count.	Per proxy server	CountStatistic	Basic	Low	33
CacheEsiInvalidate Count	CacheEsiInvalidate Count	The total ESI cache invalidate count.	Per proxy server	CountStatistic	Basic	Low	34
CacheEsiEdge CacheableCount	CacheEsiEdge CacheableCount	The number of responses that are ESI edge cacheable.	Per proxy server	CountStatistic	Extended	Low	35

CacheEsiEdge CachedCount	CacheEsiEdge CachedCount	The number of ESI edge cacheable responses that are really cached in the proxy server	Per proxy server	CountStatistic	Extended	Low	36
ActiveService ContextCount	ActiveService ContextCount	The number of active service contexts in the proxy at a given time.	Per proxy server	CountStatistic	Extended	High	41
SuspendedService ContextCount	SuspendedService ContextCount	The number of suspended service context in the proxy at a given time.	Per proxy server	CountStatistic	Extended	High	42

Related tasks

Chapter 3, “Monitoring overall system health,” on page 5
Monitoring overall system health is fundamentally important to understanding the health of every system involved with your system. This includes Web servers, application servers, databases, back-end systems, and any other systems critical to running your Web site.

Monitoring the proxy server with PMI
You can monitor the traffic of a proxy server using the Performance Monitoring Infrastructure (PMI) function.

Related reference

“PMI data organization” on page 9
Use this page as a general overview of monitoring, data collection, and counters using Performance Monitoring Infrastructure (PMI) and Tivoli Performance Viewer (TPV).

Service integration bus counters

These service integration bus counters are part of the performance monitoring infrastructure (PMI), which provides server-side monitoring and a client-side API to retrieve performance information.

To see lists of available PMI counters, use the administrative console to navigate to **Monitoring and Tuning** → **Performance Monitoring Infrastructure (PMI)** → ***resource_name*** → **[General Properties] Custom**.

For information on service integration bus PMI counters, see the following articles:

- “Message store counters”
- “Mediation framework counters” on page 92
- “Message processor counters” on page 95
- “Communications counters” on page 108
- “Web services gateway counters” on page 135

Message store counters:

These service integration bus message store counters are part of the performance monitoring infrastructure (PMI), which provides server-side monitoring and a client-side API to retrieve performance information.

Counter definitions*:* To see lists of available PMI counters, use the administrative console to navigate to **Monitoring and Tuning** → **Performance Monitoring Infrastructure (PMI)** → ***resource_name*** → **[General Properties] Custom**.

resource_name → SIB Service → SIB Messaging Engines → *bus_name* → Storage Management → Cache

Name	Key	ID	Description	Granularity	Type	Level
CacheAddNotStoredCount	MessageStoreStats.CacheAddNotStoredCount	1502	The number of items that have been added to the message store during the current session that are not persistent	Per Messaging Engine	CountStatistic	High
CacheAddStoredCount	MessageStoreStats.CacheAddStoredCount	1501	The number of items that have been added to the message store during the current session that are either persistent or potentially persistent	Per Messaging Engine	CountStatistic	High
CacheCurrentNotStoredByteCount	MessageStoreStats.CacheCurrentNotStoredByteCount	1511	The current total of the declared sizes of all items in the dynamic memory cache which are never persisted	Per Messaging Engine	CountStatistic	High
CacheCurrentNotStoredCount	MessageStoreStats.CacheCurrentNotStoredCount	1509	The number of items currently in the dynamic memory cache which are never persisted	Per Messaging Engine	CountStatistic	High
CacheCurrentStoredByteCount	MessageStoreStats.CacheCurrentStoredByteCount	1510	The total of the declared sizes of all items currently in the dynamic memory cache which are either persistent or potentially persistent	Per Messaging Engine	CountStatistic	High
CacheCurrentStoredCount	MessageStoreStats.CacheCurrentStoredCount	1508	The number of items currently in the dynamic memory cache which are either persistent or potentially persistent	Per Messaging Engine	CountStatistic	High
CacheNotStoredDiscardByteCount	MessageStoreStats.CacheNotStoredDiscardByteCount	1519	The total of the declared sizes of all items which have been added to the dynamic memory cache during the current session which are never persisted	Per Messaging Engine	CountStatistic	High
CacheNotStoredDiscardCount	MessageStoreStats.CacheNotStoredDiscardCount	1517	The total number of items which have been discarded from the dynamic memory cache during the current session which are never persisted	Per Messaging Engine	CountStatistic	High
CacheNotStoredRefusalCount	MessageStoreStats.CacheNotStoredRefusalCount	1521	The total number of items which have been refused entry to the dynamic memory cache during the current session which are never persisted	Per Messaging Engine	CountStatistic	High
CacheRemoveNotStoredCount	MessageStoreStats.CacheRemoveNotStoredCount	1506	The number of items that have been removed from the message store during the current session that are not persistent	Per Messaging Engine	CountStatistic	High
CacheRemoveStoredCount	MessageStoreStats.CacheRemoveStoredCount	1505	The number of items that have been removed from the message store during the current session that are either persistent or potentially persistent	Per Messaging Engine	CountStatistic	High
CacheRestoreCount	MessageStoreStats.CacheRestoreCount	1507	The number of items restored to memory from persistence during the current session	Per Messaging Engine	CountStatistic	High

CacheStoredDiscardByteCount	MessageStoreStats.CacheStoredDiscardByteCount	1518	The total of the declared sizes of all items which have been added to the dynamic memory cache during the current session which are either persistent or potentially persistent	Per Messaging Engine	CountStatistic	High
CacheStoredDiscardCount	MessageStoreStats.CacheStoredDiscardCount	1516	The total number of items which have been discarded from the dynamic memory cache during the current session which are either persistent or potentially persistent	Per Messaging Engine	CountStatistic	High
CacheStoredRefusalCount	MessageStoreStats.CacheStoredRefusalCount	1520	The total number of items which have been refused entry to the dynamic memory cache during the current session which are either persistent or potentially persistent	Per Messaging Engine	CountStatistic	High
CacheStreamSpillingCount	MessageStoreStats.CacheStreamSpillingCount	1522	Number of streams currently spilling potentially persistent items	Per Messaging Engine	CountStatistic	High
CacheTotalNotStoredByteCount	MessageStoreStats.CacheTotalNotStoredByteCount	1515	The total of the declared sizes of all items which have been added to the dynamic memory cache during the current session which are never persisted in cache	Per Messaging Engine	CountStatistic	High
CacheTotalNotStoredCount	MessageStoreStats.CacheTotalNotStoredCount	1513	The total number of items which have been added to the dynamic memory cache during the current session which are never persisted in cache	Per Messaging Engine	CountStatistic	High
CacheTotalStoredByteCount	MessageStoreStats.CacheTotalStoredByteCount	1514	The total of the declared sizes of all items which have been added to the dynamic memory cache during the current session which are either persistent or potentially persistent	Per Messaging Engine	CountStatistic	High
CacheTotalStoredCount	MessageStoreStats.CacheTotalStoredCount	1512	The total number of items which have been added to the dynamic memory cache during the current session which are either persistent or potentially persistent	Per Messaging Engine	CountStatistic	High
CacheUpdateNotStoredCount	MessageStoreStats.CacheUpdateNotStoredCount	1504	The number of items that have been updated in the message store during the current session that not persistent	Per Messaging Engine	CountStatistic	High
CacheUpdateStoredCount	MessageStoreStats.CacheUpdateStoredCount	1503	The number of items that have been updated in the message store during the current session that are either persistent or potentially persistent	Per Messaging Engine	CountStatistic	High

resource_name → **SIB Service** → **SIB Messaging Engines** → ***bus_name*** → **Storage Management** → **Expiry**

Name	Key	ID	Description	Granularity	Type	Level
ExpiryIndexItemCount	MessageStoreStats.ExpiryIndexItemCount	1532	Current number of items in the expiry index. These are items created with an expiry time in the future and which have not yet been consumed.	Per Messaging Engine	CountStatistic	High

resource_name → **SIB Service** → **SIB Messaging Engines** → ***bus_name*** → **Storage Management** → **Data Store**

Name	Key	ID	Description	Granularity	Type	Level
ItemDeleteBatchCount	MessageStoreStats.ItemDeleteBatchCount	1548	Item table delete batches	Per Messaging Engine	CountStatistic	High
ItemInsertBatchCount	MessageStoreStats.ItemInsertBatchCount	1547	Item table insert batches	Per Messaging Engine	CountStatistic	High
ItemUpdateBatchCount	MessageStoreStats.ItemUpdateBatchCount	1549	Item table update batches	Per Messaging Engine	CountStatistic	High
JDBCItemDeleteCount	MessageStoreStats.JDBCItemDeleteCount	1545	JDBC Item table deletes	Per Messaging Engine	CountStatistic	High
JDBCItemInsertCount	MessageStoreStats.JDBCItemInsertCount	1544	JDBC Item table inserts	Per Messaging Engine	CountStatistic	High
JDBCItemUpdateCount	MessageStoreStats.JDBCItemUpdateCount	1546	JDBC Item table updates	Per Messaging Engine	CountStatistic	High
JDBCOpenCount	MessageStoreStats.JDBCOpenCount	1540	JDBC connections open	Per Messaging Engine	CountStatistic	High
JDBCTransactionAbortCount	MessageStoreStats.JDBCTransactionAbortCount	1542	JDBC local transactions aborted	Per Messaging Engine	CountStatistic	High
JDBCTransactionCompleteCount	MessageStoreStats.JDBCTransactionCompleteCount	1541	JDBC local transactions completed	Per Messaging Engine	CountStatistic	High
JDBCTransactionDeleteCount	MessageStoreStats.JDBCTransactionDeleteCount	1551	JDBC transaction table deletes	Per Messaging Engine	CountStatistic	High
JDBCTransactionInsertCount	MessageStoreStats.JDBCTransactionInsertCount	1550	JDBC transaction table inserts	Per Messaging Engine	CountStatistic	High
JDBCTransactionTime	MessageStoreStats.JDBCTransactionTime	1543	Total execution time of internal batches	Per Messaging Engine	TimeStatistic	High
JDBCTransactionUpdateCount	MessageStoreStats.JDBCTransactionUpdateCount	1552	JDBC transaction table updates	Per Messaging Engine	CountStatistic	High
PersistentDispatcherAvoidanceCount	MessageStoreStats.PersistentDispatcherAvoidanceCount	1530	Measures the number of operations on reliable persistent data dispatched for writing to the data store but whose writing was subsequently unnecessary.	Per Messaging Engine	AverageStatistic	High

resource_name → **SIB Service** → **SIB Messaging Engines** → ***bus_name*** → **Storage Management** → **Data Store**

PersistentDispatcherAvoidanceSize	MessageStoreStats.PersistentDispatcherAvoidanceSize	1531	Measures the number of bytes associated with operations on reliable persistent data which were dispatched for writing to the data store but whose writing was subsequently unnecessary.	Per Messaging Engine	AverageStatistic	High
PersistentDispatcherBatchSize	MessageStoreStats.PersistentDispatcherBatchSize	1528	Measures the batching of operations on reliable persistent data dispatched for writing to the data store.	Per Messaging Engine	AverageStatistic	High
PersistentDispatcherCancellationCount	MessageStoreStats.PersistentDispatcherCancellationCount	1529	Counts the number of global transaction completion phases whose operations cancelled out before being written to the data store.	Per Messaging Engine	AverageStatistic	High
PersistentDispatcherRequestSize	MessageStoreStats.PersistentDispatcherRequestSize	1527	Measures the number of operations on reliable persistent data dispatched for writing to the data store.	Per Messaging Engine	AverageStatistic	High
SpillDispatcherAvoidanceCount	MessageStoreStats.SpillDispatcherAvoidanceCount	1525	Measures the number of operations on nonpersistent data dispatched for spilling to the data store but whose spilling was subsequently unnecessary.	Per Messaging Engine	AverageStatistic	High
SpillDispatcherAvoidanceSize	MessageStoreStats.SpillDispatcherAvoidanceSize	1526	Measures the number of bytes associated with operations on nonpersistent data dispatched for spilling to the data store but whose spilling was subsequently unnecessary.	Per Messaging Engine	AverageStatistic	High
SpillDispatcherBatchSize	MessageStoreStats.SpillDispatcherBatchSize	1524	Measures the batching of operations on nonpersistent data dispatched for spilling to the data store.	Per Messaging Engine	AverageStatistic	High
SpillDispatcherRequestSize	MessageStoreStats.SpillDispatcherRequestSize	1523	Measures the number of operations on nonpersistent data dispatched for spilling to the data store.	Per Messaging Engine	AverageStatistic	High
TransactionDeleteBatchCount	MessageStoreStats.TransactionDeleteBatchCount	1554	Transaction table delete batches	Per Messaging Engine	CountStatistic	High
TransactionInsertBatchCount	MessageStoreStats.TransactionInsertBatchCount	1553	Transaction table insert batches	Per Messaging Engine	CountStatistic	High

resource_name → **SIB Service** → **SIB Messaging Engines** → ***bus_name*** → **Storage Management** → **Data Store**

TransactionUpdateBatchCount	MessageStoreStats.TransactionUpdateBatchCount	1555	Transaction table update batches	Per Messaging Engine	CountStatistic	High

resource_name → **SIB Service** → **SIB Messaging Engines** → ***bus_name*** → **Storage Management** → **File Store**

Name	Key	ID	Description	Granularity	Type	Level
FileStoreLogSpace	MessageStoreStats.FileStoreLogSpace	1556	Space in bytes left in the file store log.	Per Messaging Engine	AverageStatistic	High
FileStorePermanentObjectStoreSpace	MessageStoreStats.FileStorePermanentObjectStoreSpace	1557	Space in bytes left in the file store permanent store.	Per Messaging Engine	AverageStatistic	High
FileStoreTemporaryObjectStoreSpace	MessageStoreStats.FileStoreTemporaryObjectStoreSpace	1558	Space in bytes left in the file store temporary store.	Per Messaging Engine	AverageStatistic	High

resource_name → **SIB Service** → **SIB Messaging Engines** → ***bus_name*** → **Storage Management** → **Transactions**

Name	Key	ID	Description	Granularity	Type	Level
GlobalTransactionAbortCount	MessageStoreStats.GlobalTransactionAbortCount	1538	Global transactions cancelled	Per Messaging Engine	CountStatistic	High
GlobalTransactionCommitCount	MessageStoreStats.GlobalTransactionCommitCount	1539	Global transactions committed	Per Messaging Engine	CountStatistic	High
GlobalTransactionInDoubtCount	MessageStoreStats.GlobalTransactionInDoubtCount	1537	Global transactions in doubt	Per Messaging Engine	CountStatistic	High
GlobalTransactionStartCount	MessageStoreStats.GlobalTransactionStartCount	1536	Global transactions started	Per Messaging Engine	CountStatistic	High
LocalTransactionAbortCount	MessageStoreStats.LocalTransactionAbortCount	1534	Local transactions aborted	Per Messaging Engine	CountStatistic	High
LocalTransactionCommitCount	MessageStoreStats.LocalTransactionCommitCount	1535	Local transactions committed	Per Messaging Engine	CountStatistic	High
LocalTransactionStartCount	MessageStoreStats.LocalTransactionStartCount	1533	Local transactions started	Per Messaging Engine	CountStatistic	High

Mediation framework counters:

These service integration bus mediation framework counters are part of the performance monitoring infrastructure (PMI), which provides server-side monitoring and a client-side API to retrieve performance information.

Counter definitions*:* To see lists of available PMI counters, use the administrative console to navigate to **Monitoring and Tuning** → **Performance Monitoring Infrastructure (PMI)** → ***resource_name*** → **[General Properties] Custom**.

resource_name → **SIB Service** → **SIB Messaging Engines** → ***bus_name*** → **Mediations** → **Mediation** → **Thread Pool**

Name	Key	ID	Description	Granularity	Type	Level
ThreadCount	Mediation.ThreadCount	1001	The number of messages being mediated concurrently at a mediation.	per mediation	RangeStatistic	High

resource_name → **SIB Service** → **SIB Messaging Engines** → ***bus_name*** → **Mediations** → **Mediation** → **Destinations** → **Destination**

Name	Key	ID	Description	Granularity	Type	Level
MediatedMessagesCount	Mediation.MediatedMessageCount	1002	The number of messages that have been mediated at a mediated destination.	per mediated destination	CountStatistic	Low
MediationTime	Mediation.MediationTime	1003	The amount of time in milliseconds taken to mediate a message at a mediated destination.	per mediated destination	TimeStatistic	Low

Message processor counters:

These service integration bus message processor counters are part of the performance monitoring infrastructure (PMI), which provides server-side monitoring and a client-side API to retrieve performance information.

Counter definitions*:* To see lists of available PMI counters, use the administrative console to navigate to **Monitoring and Tuning** → **Performance Monitoring Infrastructure (PMI)** → ***resource_name*** → **[General Properties] Custom**.

resource_name → **SIB Service** → **SIB Messaging Engines** → *bus_name* → **Destinations** → **Queues**

Name	Key	ID	Description	Granularity	Type	Level
AggregateMessageWaitTime	QueueStats.AggregateMessageWaitTime	18	The time spent by messages in the bus at consumption. If this time is not what was expected then view the message via the admin console to decide what action needs to be taken.	Per destination	CountStatistic	Low
AssuredPersistentMessagesConsumedCount	QueueStats.AssuredPersistentMessagesConsumedCount	16	The number of Assured Persistent messages consumed, for the lifetime of this messaging engine.	Per destination	CountStatistic	Low
AssuredPersistentMessagesProducedCount	QueueStats.AssuredPersistentMessagesProducedCount	10	The number of Assured Persistent messages produced, for the lifetime of this messaging engine.	Per destination	CountStatistic	Low
AvailableMessageCount	QueueStats.AvailableMessageCount	21	The number of messages available for a queue for consumption. If this number is close to the destination high messages value then review the high messages value.	Per destination	CountStatistic	Low
BestEffortNonPersistentMessagesConsumedCount	QueueStats.BestEffortNonPersistentMessagesConsumedCount	12	The number of Best Effort Non-persistent messages consumed, for the lifetime of this messaging engine.	Per destination	CountStatistic	Low

resource_name → **SIB Service** → **SIB Messaging Engines** → ***bus_name*** → **Destinations** → **Queues**

BestEffortNonPersistentMessagesProducedCount	QueueStats.BestEffortNonPersistentMessagesProducedCount	6	The number of Best Effort Non-persistent messages produced, for the lifetime of this messaging engine.	Per destination	CountStatistic	Low
ExpressNonPersistentMessagesConsumedCount	QueueStats.ExpressNonPersistentMessagesConsumedCount	13	The number of Express Non-persistent messages consumed, for the lifetime of this messaging engine.	Per destination	CountStatistic	Low
ExpressNonPersistentMessagesProducedCount	QueueStats.ExpressNonPersistentMessagesProducedCount	7	The number of Express Non-persistent messages produced, for the lifetime of this messaging engine.	Per destination	CountStatistic	Low
LocalConsumerAttachesCount	QueueStats.LocalConsumerAttachesCount	3	The number of times an attachment has been made to this queue by local consumers. The lifetime of this value is the lifetime of the messaging engine.	Per destination	CountStatistic	Low
LocalConsumerCount	QueueStats.LocalConsumerCount	4	The number of currently attached local consumers.	Per destination	CountStatistic	Low

resource_name → **SIB Service** → **SIB Messaging Engines** → ***bus_name*** → **Destinations** → **Queues**

LocalMessageWaitTime	QueueStats.LocalMessageWaitTime	19	The time spent by messages on this queue at consumption. If this time is not what was expected then view the message via the admin console to decide what action needs to be taken.	Per destination	CountStatistic	Low
LocalOldestMessageAge	QueueStats.LocalOldestMessageAge	20	The longest time any message has spent on this queue. If this time is not what was expected then view the message via the admin console to decide what action needs to be taken.	Per destination	CountStatistic	Low
LocalProducerAttachesCount	QueueStats.LocalProducerAttachesCount	1	The number of times an attachment has been made to this queue by local producers. The lifetime of this value is the lifetime of the messaging engine.	Per destination	CountStatistic	Low
LocalProducerCount	QueueStats.LocalProducerCount	2	The number of currently attached local producers.	Per destination	CountStatistic	Low
ReliableNonPersistentMessagesConsumedCount	QueueStats.ReliableNonPersistentMessagesConsumedCount	14	The number of Reliable Non-persistent messages consumed, for the lifetime of this messaging engine.	Per destination	CountStatistic	Low

resource_name → **SIB Service** → **SIB Messaging Engines** → ***bus_name*** → **Destinations** → **Queues**

ReliableNonPersistentMessagesProducedCount	QueueStats.ReliableNonPersistentMessagesProducedCount	8	The number of Reliable Non-persistent messages produced, for the lifetime of this messaging engine.	Per destination	CountStatistic	Low
ReliablePersistentMessagesConsumedCount	QueueStats.ReliablePersistentMessagesConsumedCount	15	The number of Reliable Persistent messages consumed, for the lifetime of this messaging engine.	Per destination	CountStatistic	Low
ReliablePersistentMessagesProducedCount	QueueStats.ReliablePersistentMessagesProducedCount	9	The number of Reliable Persistent messages produced, for the lifetime of this messaging engine.	Per destination	CountStatistic	Low
ReportEnabledMessagesExpiredCount	QueueStats.ReportEnabledMessagesExpiredCount	17	The number of report enabled messages that expired while on this queue.	Per destination	CountStatistic	Low
TotalMessagesConsumedCount	QueueStats.TotalMessagesConsumedCount	11	The total number of messages consumed from this queue, for the lifetime of this messaging engine.	Per destination	CountStatistic	Low
TotalMessagesProducedCount	QueueStats.TotalMessagesProducedCount	5	The total number of messages produced to this queue, for the lifetime of this messaging engine.	Per destination	CountStatistic	Low

resource_name → **SIB Service** → **SIB Messaging Engines** → ***bus_name*** → **Destinations** → **Queues**

UnavailableMessageCount	QueueStats.UnavailableMessageCount	22	The number of messages locked or uncommitted, this means messages that have been added or removed but the transaction has not been committed yet. If this number is high then check which messages are locked and why.	Per destination	CountStatistic	Low

resource_name → **SIB Service** → **SIB Messaging Engines** → ***bus_name*** → **Destinations** → **Topicspaces**

Name	Key	ID	Description	Granularity	Type	Level
AssuredPersistentLocalSubscriptionHitCount	TopicspaceStats.AssuredPersistentLocalSubscriptionHitCount	116	The cumulative total of subscriptions which have matched Assured Persistent publications.	Per destination	CountStatistic	Low
AssuredPersistentMessagesPublishedCount	TopicspaceStats.AssuredPersistentMessagesPublishedCount	110	The number of Assured Persistent messages published	Per destination	CountStatistic	Low
BestEffortNonPersistentLocalSubscriptionHitCount	TopicspaceStats.BestEffortNonPersistentLocalSubscriptionHitCount	112	The cumulative total of subscriptions which have matched Best Effort Non-persistent publications.	Per destination	CountStatistic	Low
BestEffortNonPersistentMessagesPublishedCount	TopicspaceStats.BestEffortNonPersistentMessagesPublishedCount	106	The number of Best Effort Non-persistent messages published	Per destination	CountStatistic	Low
DurableLocalSubscriptionCount	TopicspaceStats.DurableLocalSubscriptionCount	104	The number of durable subscriptions.	Per destination	CountStatistic	Low
ExpressNonPersistentLocal SubscriptionHitCount	TopicspaceStats.ExpressNonPersistentLocalSubscriptionHitCount	113	The cumulative total of subscriptions which have matched Express Non-persistent publications.	Per destination	CountStatistic	Low
ExpressNonPersistentMessagesPublishedCount	TopicspaceStats.ExpressNonPersistentMessagesPublishedCount	107	The number of Express Non-persistent messages published	Per destination	CountStatistic	Low

resource_name → **SIB Service** → **SIB Messaging Engines** → ***bus_name*** → **Destinations** → **Topicspaces**

IncompletePublicationCount	TopicspaceStats.IncompletePublicationCount	119	The number of publications not yet received by all current subscribers. If this number is unexpected then view the publication via the admin console to take any actions.	Per destination	CountStatistic	Low
LocalOldestPublicationAge	TopicspaceStats.LocalOldestPublicationAge	118	The longest time any publication has spent on this topicspace. If this time is not what was expected then view the message via the admin console to decide what action needs to be taken.	Per destination	TimeStatistic	max
LocalPublisherAttachesCount	TopicspaceStats.LocalPublisherAttachesCount	100	The number of times an attachment has been made to this topicspace by local producers. The lifetime of this value is the lifetime of the messaging engine.	Per destination	CountStatistic	Low
LocalPublisherCount	TopicspaceStats.LocalPublisherCount	101	The number of local publishers to topics in this topicspace.	Per destination	CountStatistic	Low
NonDurableLocalSubscriptionCount	TopicspaceStats.NonDurableLocalSubscriptionCount	103	The number of non-durable subscriptions.	Per destination	CountStatistic	Low

resource_name → **SIB Service** → **SIB Messaging Engines** → ***bus_name*** → **Destinations** → **Topicspaces**

ReliableNonPersistentLocalSubscriptionHitCount	TopicspaceStats.ReliableNonPersistentLocalSubscriptionHitCount	114	The cumulative total of subscriptions which have matched Reliable Non-persistent publications.	Per destination	CountStatistic	Low
ReliableNonPersistentMessagesPublishedCount	TopicspaceStats.ReliableNonPersistentMessagesPublishedCount	108	The number of Reliable Non-persistent messages published	Per destination	CountStatistic	Low
ReliablePersistentLocalSubscriptionHitCount	TopicspaceStats.ReliablePersistentLocalSubscriptionHitCount	115	The cumulative total of subscriptions which have matched Reliable Persistent publications.	Per destination	CountStatistic	Low
ReliablePersistentMessagesPublishedCount	TopicspaceStats.ReliablePersistentMessagesPublishedCount	109	The number of Reliable Persistent messages published	Per destination	CountStatistic	Low
ReportEnabledPublicationExpiredCount	TopicspaceStats.ReportEnabledPublicationsExpiredCount	117	The number of report enabled incomplete publications that expired while on this topicspace.	Per destination	CountStatistic	Low
TotalLocalSubscriptionCount	TopicspaceStats.TotalLocalSubscriptionCount	102	The number of local subscriptions to topics in this topicspace. Each subscription is counted once, even if the topic includes wildcards.	Per destination	CountStatistic	Low
TotalLocalSubscriptionHitCount	TopicspaceStats.TotalLocalSubscriptionHitCount	111	The cumulative total of subscriptions which have matched topicspace publications.	Per destination	CountStatistic	Low

resource_name → **SIB Service** → **SIB Messaging Engines** → ***bus_name*** → **Destinations** → **Topicspaces**

TotalMessagesPublishedCount	TopicspaceStats.TotalMessagesPublishedCount	105	The total number of publications to this topicspace.	Per destination	CountStatistic	Low

resource_name → **SIB Service** → **SIB Messaging Engines** → ***bus_name*** → **Destinations** → **Topicspaces** → **Topic** → **Durable Subscriptions**

Name	Key	ID	Description	Granularity	Type	Level
AggregateMessageWaitTime	DurableSubscriptionStats.AggregateMessageWaitTime	206	The time spent by messages in the bus at consumption. If this time is not what was expected then view the message via the admin console to decide what action needs to be taken.	per mediated destination	TimeStatistic	High
AssuredPersistentMessagesConsumedCount	DurableSubscriptionStats.AssuredPersistentMessagesConsumedCount	205	The number of Assured Persistent messages consumed, for the lifetime of this messaging engine.	per mediated destination	CountStatistic	Low
AvailabeMessageCount	DurableSubscriptionStats.AvailableMessageCount	209	The number of messages waiting to be consumed.	per mediated destination	CountStatistic	Low
BestEffortNonPersistentMessagesConsumedCount	DurableSubscriptionStats.BestEffortNonPersistentMessagesConsumedCount	201	The number of Best Effort Non-persistent messages consumed, for the lifetime of this messaging engine.	per mediated destination	CountStatistic	Low
ExpressNonPersistentMessagesConsumedCount	DurableSubscriptionStats.ExpressNonPersistentMessagesConsumedCount	202	The number of Express Non-persistent messages consumed, for the lifetime of this messaging engine.	per mediated destination	CountStatistic	Low

resource_name → **SIB Service** → **SIB Messaging Engines** → ***bus_name*** → **Destinations** → **Topicspaces** → **Topic** → **Durable Subscriptions**

LocalMessageWaitTime	DurableSubscriptionStats.LocalMessageWaitTime	207	The time spent by messages on this durable subscription at consumption. If this time is not what was expected then view the message via the admin console to decide what action needs to be taken.	per mediated destination	TimeStatistic	High
LocalOldestPublicationAge	DurableSubscriptionStats.LocalOldestPublicationAge	208	The longest time any message has spent on this subscription. If this time is not what was expected then view the message via the admin console to decide what action needs to be taken.	per mediated destination	TimeStatistic	Max
ReliableNonPersistentMessagesConsumedCount	DurableSubscriptionStats.ReliableNonPersistentMessagesConsumedCount	203	The number of Reliable Non-persistent messages consumed, for the lifetime of this messaging engine.	per mediated destination	CountStatistic	Low
ReliablePersistentMessagesConsumedCount	DurableSubscriptionStats.ReliablePersistentMessagesConsumedCount	204	The number of Reliable Persistent messages consumed, for the lifetime of this messaging engine.	per mediated destination	CountStatistic	Low
TotalMessagesConsumedCount	DurableSubscriptionStats.TotalMessagesConsumedCount	200	The total number of messages consumed from this durable subscription.	per mediated destination	CountStatistic	Low

Communications counters:

These service integration bus communications counters are part of the performance monitoring infrastructure (PMI), which provides server-side monitoring and a client-side API to retrieve performance information. Communications counters are maintained across all components of WebSphere Application Server.

Counter definitions*:* To see lists of available PMI counters, use the administrative console to navigate to **Monitoring and Tuning** → **Performance Monitoring Infrastructure (PMI)** → ***resource_name*** → **[General Properties] Custom**.

resource_name → **SIB Service** → **SIB Communications** → **Clients** → **Standard Statistics**

Name	Key	ID	Description	Granularity	Type	Level
APIConnectionsCount	ClientStats.APIConnections	552	The number of API sessions being used by clients that are currently network connected to this application server. Some of these API connections might be being by internal system processes on behalf of a client.	current clients/connections connected to this application server	CountStatistic	Low
BufferedReadBytesCount	ClientStats.BufferedReadBytes	561	Number of bytes of data that have been received from the network and are held pending further processing. Large values might indicate that the application server is unable to process data fast enough to keep up with the clients attached.	All clients connected or that have been connected to this application server	CountStatistic	Low
BufferedWriteBytesCount	ClientStats.BufferedWriteBytes	560	Number of bytes of data being held pending transmission. Large values might indicate network congestion or clients which are unable to process data fast enough to keep up with the application server.	All clients connected or that have been connected to this application server	CountStatistic	Low
ClientsAttachedCount	ClientStats.ClientsAttached	551	The number of distinct client processes currently network connected to this application server.	current clients/connections connected to this application server	CountStatistic	Low
ErrorsCount	ClientStats.Errors	553	Communication errors that have occurred and resulted in a network connection to a client being disconnected.	All clients connected or that have been connected to this application server	CountStatistic	Low
MessageBytesReadCount	ClientStats.MessageBytesRead	563	Number of bytes of message data received from client processes over network connections. This does not include data used to negotiate the transmission of messages	All clients connected or that have been connected to this application server	CountStatistic	Low
MessagesBytesWrittenCount	ClientStats.MessageBytesWritten	562	Number of bytes of message data sent to client processes over network connections. This does not include data used to negotiate the transmission of messages	All clients connected or that have been connected to this application server	CountStatistic	Low

resource_name → **SIB Service** → **SIB Communications** → **Clients** → **Standard Statistics**

MulticastSendMessageCount	ClientStats.MulticastSendMessage	559	Number of messages transmitted using multicast protocols.	All clients connected or that have been connected to this application server	CountStatistic	Low
MulticastWriteBytesCount	ClientStats.MulticastWriteBytes	558	Number of bytes transmitted using multicast protocols.	All clients connected or that have been connected to this application server	CountStatistic	Low
ReadsBlockedCount	ClientStats.ReadsBlocked	557	Number of read operations that could not be completed immediately. This number can be used as an indicator of network congestion when communicating with client processes.	All clients connected or that have been connected to this application server	CountStatistic	Low
ReadsCount	ClientStats.Reads	555	Number of read operations used to receive data from client processes via network connections.	All clients connected or that have been connected to this application server	CountStatistic	Low
TotalBytesReadCount	ClientStats.TotalBytesRead	565	Number of bytes of data received from client processes. This includes both message data and data used to negotiate the transmission of messages.	All clients connected or that have been connected to this application server	CountStatistic	Low
TotalBytesWrittenCount	ClientStats.TotalBytesWritten	564	Number of bytes of data sent to client processes. This includes both message data and data used to negotiate the transmission of messages.	All clients connected or that have been connected to this application server	CountStatistic	Low
WritesBlockedCount	ClientStats.WritesBlocked	556	Number of write operations that could not be completed immediately. This number can be used as an indicator of network congestion when communicating with client processes.	All clients connected or that have been connected to this application server	CountStatistic	Low
WritesCount	ClientStats.Writes	554	Number of write operations used to transmit data to client processes via network connections.	All clients connected or that have been connected to this application server	CountStatistic	Low

resource_name → **SIB Service** → **SIB Communications** → **Clients** → **Detailed Statistics**

Name	Key	ID	Description	Granularity	Type	Level
BytesReceivedAtHighPriorityCount	ClientDetailedStats.BytesReceivedAtHighPriority	719	Number of bytes of data received at a high priority. Message data cannot be transmitted with this priority, so typically these bytes of data will comprise control transmissions used to negotiate the flow of messages.	All clients connected or that have been connected to this application server	CountStatistic	Low
BytesReceivedAtHighestPriorityCount	ClientDetailedStats.BytesReceivedAtHighestPriority	717	Number of bytes of data received at the highest possible priority. Message data cannot be transmitted with this priority, so typically these bytes of data will comprise control transmissions used to negotiate the flow of messages.	All clients connected or that have been connected to this application server	CountStatistic	Low
BytesReceivedAtJMS0PriorityCount	ClientDetailedStats.BytesReceivedAtJMS0Priority	729	Number of bytes of data received at the priority used by JMS priority 0 messages. Typically this is an accurate measure of the number of bytes of message data received at this priority level. However, from time to time, control transmissions used to negotiate the flow of messages might be transmitted at this priority level.	All clients connected or that have been connected to this application server	CountStatistic	Low

resource_name → **SIB Service** → **SIB Communications** → **Clients** → **Detailed Statistics**

BytesReceivedAtJMS1PriorityCount	ClientDetailedStats.BytesReceivedAtJMS1Priority	728	Number of bytes of data received at the priority used by JMS priority 1 messages. Typically this is an accurate measure of the number of bytes of message data received at this priority level. However, from time to time, control transmissions used to negotiate the flow of messages might be transmitted at this priority level.	All clients connected or that have been connected to this application server	CountStatistic	Low
BytesReceivedAtJMS2PriorityCount	ClientDetailedStats.BytesReceivedAtJMS2Priority	727	Number of bytes of data received at the priority used by JMS priority 2 messages. Typically this is an accurate measure of the number of bytes of message data received at this priority level. However, from time to time, control transmissions used to negotiate the flow of messages might be transmitted at this priority level.	All clients connected or that have been connected to this application server	CountStatistic	Low

resource_name → **SIB Service** → **SIB Communications** → **Clients** → **Detailed Statistics**

BytesReceivedAtJMS3PriorityCount	ClientDetailedStats.BytesReceivedAtJMS3Priority	726	Number of bytes of data received at the priority used by JMS priority 3 messages. Typically this is an accurate measure of the number of bytes of message data received at this priority level. However, from time to time, control transmissions used to negotiate the flow of messages might be transmitted at this priority level.	All clients connected or that have been connected to this application server	CountStatistic	Low
BytesReceivedAtJMS4PriorityCount	ClientDetailedStats.BytesReceivedAtJMS4Priority	725	Number of bytes of data received at the priority used by JMS priority 4 messages. Typically this is an accurate measure of the number of bytes of message data received at this priority level. However, from time to time, control transmissions used to negotiate the flow of messages might be transmitted at this priority level.	All clients connected or that have been connected to this application server	CountStatistic	Low

resource_name → **SIB Service** → **SIB Communications** → **Clients** → **Detailed Statistics**

BytesReceivedAtJMS5PriorityCount	ClientDetailedStats.BytesReceivedAtJMS5Priority	724	Number of bytes of data received at the priority used by JMS priority 5 messages. Typically this is an accurate measure of the number of bytes of message data received at this priority level. However, from time to time, control transmissions used to negotiate the flow of messages might be transmitted at this priority level.	All clients connected or that have been connected to this application server	CountStatistic	Low
BytesReceivedAtJMS6PriorityCount	ClientDetailedStats.BytesReceivedAtJMS6Priority	723	Number of bytes of data received at the priority used by JMS priority 6 messages. Typically this is an accurate measure of the number of bytes of message data received at this priority level. However, from time to time, control transmissions used to negotiate the flow of messages might be transmitted at this priority level.	All clients connected or that have been connected to this application server	CountStatistic	Low

resource_name → **SIB Service** → **SIB Communications** → **Clients** → **Detailed Statistics**

BytesReceivedAtJMS7PriorityCount	ClientDetailedStats.BytesReceivedAtJMS7Priority	722	Number of bytes of data received at the priority used by JMS priority 7 messages. Typically this is an accurate measure of the number of bytes of message data received at this priority level. However, from time to time, control transmissions used to negotiate the flow of messages might be transmitted at this priority level.	All clients connected or that have been connected to this application server	CountStatistic	Low
BytesReceivedAtJMS8PriorityCount	ClientDetailedStats.BytesReceivedAtJMS8Priority	721	Number of bytes of data received at the priority used by JMS priority 8 messages. Typically this is an accurate measure of the number of bytes of message data received at this priority level. However, from time to time, control transmissions used to negotiate the flow of messages might be transmitted at this priority level.	All clients connected or that have been connected to this application server	CountStatistic	Low

resource_name → **SIB Service** → **SIB Communications** → **Clients** → **Detailed Statistics**

BytesReceivedAtJMS9PriorityCount	ClientDetailedStats.BytesReceivedAtJMS9Priority	720	Number of bytes of data received at the priority used by JMS priority 9 messages. Typically this is an accurate measure of the number of bytes of message data received at this priority level. However, from time to time, control transmissions used to negotiate the flow of messages might be transmitted at this priority level.	All clients connected or that have been connected to this application server	CountStatistic	Low
BytesReceivedAtLowPriorityCount	ClientDetailedStats.BytesReceivedAtLowPriority	730	Number of bytes of data received at a low priority. Message data cannot be transmitted with this priority, so typically these bytes of data will comprise control transmissions used to negotiate the flow of messages.	All clients connected or that have been connected to this application server	CountStatistic	Low
BytesReceivedAtLowestPriorityCount	ClientDetailedStats.BytesReceivedAtLowestPriority	732	Number of bytes of data received at the lowest possible priority. Message data cannot be transmitted with this priority, so typically these bytes of data will comprise control transmissions used to negotiate the flow of messages.	All clients connected or that have been connected to this application server	CountStatistic	Low

resource_name → **SIB Service** → **SIB Communications** → **Clients** → **Detailed Statistics**

BytesReceivedAtVeryHighPriorityCount	ClientDetailedStats.BytesReceivedAtVeryHighPriority	718	Number of bytes of data received at a very high priority. Message data cannot be transmitted with this priority, so typically these bytes of data will comprise control transmissions used to negotiate the flow of messages.	All clients connected or that have been connected to this application server	CountStatistic	Low
BytesReceivedAtVeryLowPriorityCount	ClientDetailedStats.BytesReceivedAtVeryLowPriority	731	Number of bytes of data received at a very low priority. Message data cannot be transmitted with this priority, so typically these bytes of data will comprise control transmissions used to negotiate the flow of messages.	All clients connected or that have been connected to this application server	CountStatistic	Low
BytesSentAtHighPriorityCount	ClientDetailedStats.BytesSentAtHighPriority	703	Number of bytes of data transmitted at a high priority. Message data cannot be transmitted with this priority, so typically these bytes of data will comprise control transmissions used to negotiate the flow of messages.	All clients connected or that have been connected to this application server	CountStatistic	Low
BytesSentAtHighestPriorityCount	ClientDetailedStats.BytesSentAtHighestPriority	701	Number of bytes of data transmitted at the highest possible priority for transmission. Message data cannot be transmitted with this priority, so typically these bytes of data will comprise control transmissions used to negotiate the flow of messages.	All clients connected or that have been connected to this application server	CountStatistic	Low

resource_name → **SIB Service** → **SIB Communications** → **Clients** → **Detailed Statistics**

BytesSentAtJMS0PriorityCount	ClientDetailedStats.BytesSentAtJMS0Priority	713	Number of bytes of data transmitted at the priority used by JMS priority 0 messages. Typically this is an accurate measure of the number of bytes of message data transmitted at this priority level. However, from time to time, control transmissions used to negotiate the flow of messages might be transmitted at this priority level.	All clients connected or that have been connected to this application server	CountStatistic	Low
BytesSentAtJMS1PriorityCount	ClientDetailedStats.BytesSentAtJMS1Priority	712	Number of bytes of data transmitted at the priority used by JMS priority 1 messages. Typically this is an accurate measure of the number of bytes of message data transmitted at this priority level. However, from time to time, control transmissions used to negotiate the flow of messages might be transmitted at this priority level.	All clients connected or that have been connected to this application server	CountStatistic	Low

resource_name → **SIB Service** → **SIB Communications** → **Clients** → **Detailed Statistics**

BytesSentAtJMS2PriorityCount	ClientDetailedStats.BytesSentAtJMS2Priority	711	Number of bytes of data transmitted at the priority used by JMS priority 2 messages. Typically this is an accurate measure of the number of bytes of message data transmitted at this priority level. However, from time to time, control transmissions used to negotiate the flow of messages might be transmitted at this priority level.	All clients connected or that have been connected to this application server	CountStatistic	Low
BytesSentAtJMS3PriorityCount	ClientDetailedStats.BytesSentAtJMS3Priority	710	Number of bytes of data transmitted at the priority used by JMS priority 3 messages. Typically this is an accurate measure of the number of bytes of message data transmitted at this priority level. However, from time to time, control transmissions used to negotiate the flow of messages might be transmitted at this priority level.	All clients connected or that have been connected to this application server	CountStatistic	Low

resource_name → **SIB Service** → **SIB Communications** → **Clients** → **Detailed Statistics**

BytesSentAtJMS4PriorityCount	ClientDetailedStats.BytesSentAtJMS4Priority	709	Number of bytes of data transmitted at the priority used by JMS priority 4 messages. Typically this is an accurate measure of the number of bytes of message data transmitted at this priority level. However, from time to time, control transmissions used to negotiate the flow of messages might be transmitted at this priority level.	All clients connected or that have been connected to this application server	CountStatistic	Low
BytesSentAtJMS5PriorityCount	ClientDetailedStats.BytesSentAtJMS5Priority	708	Number of bytes of data transmitted at the priority used by JMS priority 5 messages. Typically this is an accurate measure of the number of bytes of message data transmitted at this priority level. However, from time to time, control transmissions used to negotiate the flow of messages might be transmitted at this priority level.	All clients connected or that have been connected to this application server	CountStatistic	Low

resource_name → **SIB Service** → **SIB Communications** → **Clients** → **Detailed Statistics**

BytesSentAtJMS6PriorityCount	ClientDetailedStats.BytesSentAtJMS6Priority	707	Number of bytes of data transmitted at the priority used by JMS priority 6 messages. Typically this is an accurate measure of the number of bytes of message data transmitted at this priority level. However, from time to time, control transmissions used to negotiate the flow of messages might be transmitted at this priority level.	All clients connected or that have been connected to this application server	CountStatistic	Low
BytesSentAtJMS7PriorityCount	ClientDetailedStats.BytesSentAtJMS7Priority	706	Number of bytes of data transmitted at the priority used by JMS priority 7 messages. Typically this is an accurate measure of the number of bytes of message data transmitted at this priority level. However, from time to time, control transmissions used to negotiate the flow of messages might be transmitted at this priority level.	All clients connected or that have been connected to this application server	CountStatistic	Low

resource_name → **SIB Service** → **SIB Communications** → **Clients** → **Detailed Statistics**

BytesSentAtJMS8PriorityCount	ClientDetailedStats.BytesSentAtJMS8Priority	705	Number of bytes of data transmitted at the priority used by JMS priority 8 messages. Typically this is an accurate measure of the number of bytes of message data transmitted at this priority level. However, from time to time, control transmissions used to negotiate the flow of messages might be transmitted at this priority level.	All clients connected or that have been connected to this application server	CountStatistic	Low
BytesSentAtJMS9PriorityCount	ClientDetailedStats.BytesSentAtJMS9Priority	704	Number of bytes of data transmitted at the priority used by JMS priority 9 messages. Typically this is an accurate measure of the number of bytes of message data transmitted at this priority level. However, from time to time, control transmissions used to negotiate the flow of messages might be transmitted at this priority level.	All clients connected or that have been connected to this application server	CountStatistic	Low
BytesSentAtLowPriorityCount	ClientDetailedStats.BytesSentAtLowPriority	714	Number of bytes of data transmitted at a low priority. Message data cannot be transmitted with this priority, so typically these bytes of data will comprise control transmissions used to negotiate the flow of messages.	All clients connected or that have been connected to this application server	CountStatistic	Low

resource_name → **SIB Service** → **SIB Communications** → **Clients** → **Detailed Statistics**

BytesSentAtLowestPriorityCount	ClientDetailedStats.BytesSentAtLowestPriority	716	Number of bytes of data transmitted at the lowest possible priority. Message data cannot be transmitted with this priority, so typically these bytes of data will comprise control transmissions used to negotiate the flow of messages.	All clients connected or that have been connected to this application server	CountStatistic	Low
BytesSentAtVeryHighPriorityCount	ClientDetailedStats.BytesSentAtVeryHighPriority	702	Number of bytes of data transmitted at a very high priority. Message data cannot be transmitted with this priority, so typically these bytes of data will comprise control transmissions used to negotiate the flow of messages.	All clients connected or that have been connected to this application server	CountStatistic	Low
BytesSentAtVeryLowPriorityCount	ClientDetailedStats.BytesSentAtVeryLowPriority	715	Number of bytes of data transmitted at a very low priority. Message data cannot be transmitted with this priority, so typically these bytes of data will comprise control transmissions used to negotiate the flow of messages.	All clients connected or that have been connected to this application server	CountStatistic	Low
MessagesReceivedAtJMS0PriorityCount	ClientDetailedStats.MessagesReceivedAtJMS0Priority	752	Number of messages received at JMS priority 0.	All clients connected or that have been connected to this application server	CountStatistic	Low
MessagesReceivedAtJMS1PriorityCount	ClientDetailedStats.MessagesReceivedAtJMS1Priority	751	Number of messages received at JMS priority 1.	All clients connected or that have been connected to this application server	CountStatistic	Low

resource_name → **SIB Service** → **SIB Communications** → **Clients** → **Detailed Statistics**

MessagesReceivedAtJMS2PriorityCount	ClientDetailedStats.MessagesReceivedAtJMS2Priority	750	Number of messages received at JMS priority 2.	All clients connected or that have been connected to this application server	CountStatistic	Low
MessagesReceivedAtJMS3PriorityCount	ClientDetailedStats.MessagesReceivedAtJMS3Priority	749	Number of messages received at JMS priority 3.	All clients connected or that have been connected to this application server	CountStatistic	Low
MessagesReceivedAtJMS4PriorityCount	ClientDetailedStats.MessagesReceivedAtJMS4Priority	748	Number of messages received at JMS priority 4.	All clients connected or that have been connected to this application server	CountStatistic	Low
MessagesReceivedAtJMS5PriorityCount	ClientDetailedStats.MessagesReceivedAtJMS5Priority	747	Number of messages received at JMS priority 5.	All clients connected or that have been connected to this application server	CountStatistic	Low
MessagesReceivedAtJMS6PriorityCount	ClientDetailedStats.MessagesReceivedAtJMS6Priority	746	Number of messages received at JMS priority 6.	All clients connected or that have been connected to this application server	CountStatistic	Low
MessagesReceivedAtJMS7PriorityCount	ClientDetailedStats.MessagesReceivedAtJMS7Priority	745	Number of messages received at JMS priority 7.	All clients connected or that have been connected to this application server	CountStatistic	Low
MessagesReceivedAtJMS8PriorityCount	ClientDetailedStats.MessagesReceivedAtJMS8Priority	744	Number of messages received at JMS priority 8.	All clients connected or that have been connected to this application server	CountStatistic	Low
MessagesReceivedAtJMS9PriorityCount	ClientDetailedStats.MessagesReceivedAtJMS9Priority	743	Number of messages received at JMS priority 9.	All clients connected or that have been connected to this application server	CountStatistic	Low

resource_name → **SIB Service** → **SIB Communications** → **Clients** → **Detailed Statistics**

MessagesSentAtJMS0PriorityCount	ClientDetailedStats.MessagesSentAtJMS0Priority	742	Number of messages transmitted at JMS priority 0.	All clients connected or that have been connected to this application server	CountStatistic	Low
MessagesSentAtJMS1PriorityCount	ClientDetailedStats.MessagesSentAtJMS1Priority	741	Number of messages transmitted at JMS priority 1.	All clients connected or that have been connected to this application server	CountStatistic	Low
MessagesSentAtJMS2PriorityCount	ClientDetailedStats.MessagesSentAtJMS2Priority	740	Number of messages transmitted at JMS priority 2.	All clients connected or that have been connected to this application server	CountStatistic	Low
MessagesSentAtJMS3PriorityCount	ClientDetailedStats.MessagesSentAtJMS3Priority	739	Number of messages transmitted at JMS priority 3.	All clients connected or that have been connected to this application server	CountStatistic	Low
MessagesSentAtJMS4PriorityCount	ClientDetailedStats.MessagesSentAtJMS4Priority	738	Number of messages transmitted at JMS priority 4.	All clients connected or that have been connected to this application server	CountStatistic	Low
MessagesSentAtJMS5PriorityCount	ClientDetailedStats.MessagesSentAtJMS5Priority	737	Number of messages transmitted at JMS priority 5.	All clients connected or that have been connected to this application server	CountStatistic	Low
MessagesSentAtJMS6PriorityCount	ClientDetailedStats.MessagesSentAtJMS6Priority	736	Number of messages transmitted at JMS priority 6.	All clients connected or that have been connected to this application server	CountStatistic	Low
MessagesSentAtJMS7PriorityCount	ClientDetailedStats.MessagesSentAtJMS7Priority	735	Number of messages transmitted at JMS priority 7.	All clients connected or that have been connected to this application server	CountStatistic	Low

resource_name → **SIB Service** → **SIB Communications** → **Clients** → **Detailed Statistics**

MessagesSentAtJMS8PriorityCount	ClientDetailedStats.MessagesSentAtJMS8Priority	734	Number of messages transmitted at JMS priority 8.	All clients connected or that have been connected to this application server	CountStatistic	Low
MessagesSentAtJMS9PriorityCount	ClientDetailedStats.MessagesSentAtJMS9Priority	733	Number of messages transmitted at JMS priority 9.	All clients connected or that have been connected to this application server	CountStatistic	Low

resource_name → **SIB Service** → **SIB Communications** → **Messaging Engines** → **Standard Statistics**

Name	Key	ID	Description	Granularity	Type	Level
APIConnectionsCount	MEStats.APIConnections	502	The number of sessions being used by messaging engines that are currently network connected to this application server.	current applications servers hosting messaging engines/connections connected to this application server.	CountStatistic	Low
BufferedReadBytesCount	MEStats.BufferedReadBytes	511	Number of bytes of data that have been received from the network and are held pending further processing. Large values might indicate that the application server is unable to process data fast enough to keep up with the other application server processes hosting messaging engines that it is network attached.	All messaging engines connected or that have been connected to this application server	CountStatistic	Low
BufferedWriteBytesCount	MEStats.BufferedWriteBytes	510	Number of bytes of data being held pending transmission. Large values might indicate network congestion or application server processes hosting messaging engines which are unable to process data fast enough to keep up with the application server.	All messaging engines connected or that have been connected to this application server	CountStatistic	Low
ErrorsCount	MEStats.Errors	503	Communication errors that have occurred and resulted in a network connection to a messaging engine being disconnected.	All messaging engines connected or that have been connected to this application server	CountStatistic	Low
MEAttachedCount	MEStats.MEAttached	501	The number of distinct application server processes hosting messaging engines currently network connected to this application server.	current applications servers hosting messaging engines/connections connected to this application server.	CountStatistic	Low

resource_name → **SIB Service** → **SIB Communications** → **Messaging Engines** → **Standard Statistics**

MessageBytesReadCount	MEStats.MessageBytesRead	513	Number of bytes of message data received from application server processes hosting messaging engines over network connections. This does not include data used to negotiate the transmission of messages	All messaging engines connected or that have been connected to this application server	CountStatistic	Low
MessageBytesWrittenCount	MEStats.MessageBytesWritten	512	Number of bytes of message data sent to application server processes hosting messaging engines over network connections. This does not include data used to negotiate the transmission of messages	All messaging engines connected or that have been connected to this application server	CountStatistic	Low
ReadsBlockedCount	MEStats.ReadsBlocked	507	Number of read operations that could not be completed immediately. This number can be used as an indicator of network congestion when communicating with application server processes hosting messaging engines.	All messaging engines connected or that have been connected to this application server	CountStatistic	Low
ReadsCount	MEStats.Reads	505	Number of read operations used to receive data from application server processes hosting messaging engines via network connections.	All messaging engines connected or that have been connected to this application server	CountStatistic	Low
TotalBytesReadCount	MEStats.TotalBytesRead	515	Number of bytes of data received from application server processes hosting messaging engines.	All messaging engines connected or that have been connected to this application server	CountStatistic	Low
TotalBytesWrittenCount	MEStats.TotalBytesWritten	514	Number of bytes of data sent to application server processes hosting messaging engines.	All messaging engines connected or that have been connected to this application server	CountStatistic	Low

resource_name → **SIB Service** → **SIB Communications** → **Messaging Engines** → **Standard Statistics**

WritesBlockedCount	MEStats.WritesBlocked	506	Number of write operations that could not be completed immediately. This number can be used as an indicator of network congestion when communicating with application server processes hosting messaging engines.	All messaging engines connected or that have been connected to this application server	CountStatistic	Low
WritesCount	MEStats.Writes	504	Number of write operations used to transmit data to application server processes hosting messaging engines via network connections.	All messaging engines connected or that have been connected to this application server	CountStatistic	Low

resource_name → **SIB Service** → **SIB Communications** → **Messaging Engines** → **Detailed Statistics**

The same set of detailed statistics are available for messaging engines as is described for clients.

resource_name → **SIB Service** → **SIB Communications** → **WMQ Links** → **Standard Statistics**

Name	Key	ID	Description	Granularity	Type	Level
BatchesReceivedCount	MQLinkStats.BatchesReceived	802	Number of batches of messages received from network attached WebSphere MQ Queue Managers.	All WebSphere MQ Queue Managers that are or have been connected to this application server	CountStatistic	Low
BatchesSentCount	MQLinkStats.BatchesSent	801	Number of batches of messages sent to network attached WebSphere MQ Queue Managers.	All WebSphere MQ Queue Managers that are or have been connected to this application server	CountStatistic	Low
CommsErrorsCount	MQLinkStats.CommsErrors	811	Number of communication errors that resulted in a network connection to a WebSphere MQ Queue Manager being disconnected.	All WebSphere MQ Queue Managers that are or have been connected to this application server	CountStatistic	Low
LongRetriesCount	MQLinkStats.LongRetries	810	Number of long retries. This indicates the number of times channels were disconnected and could not be re-established for longer periods of time.	All WebSphere MQ Queue Managers that are or have been connected to this application server	CountStatistic	Low
MessagesReceivedCount	MQLinkStats.MessagesReceived	804	Number of messages received from network attached WebSphere MQ Queue Managers.	All WebSphere MQ Queue Managers that are or have been connected to this application server	CountStatistic	Low
MessagesSentCount	MQLinkStats.MessagesSent	803	Number of messages sent to network attached WebSphere MQ Queue Managers.	All WebSphere MQ Queue Managers that are or have been connected to this application server	CountStatistic	Low
QMAttachedCount	MQLinkStats.QMAttached	812	Total number of WebSphere MQ Queue Managers currently network attached to this application server	WebSphere MQ Queue Managers that are currently attached	CountStatistic	Low
ReadsBlockedCount	MQLinkStats.ReadsBlocked	814	Number of read operations that could not be completed immediately. This number can be used as an indicator of network congestion when communicating with WebSphere MQ Queue Managers.	All WebSphere MQ Queue Managers that are or have been connected to this application server	CountStatistic	Low

resource_name → **SIB Service** → **SIB Communications** → **WMQ Links** → **Standard Statistics**

ReceiverBytesReceivedCount	MQLinkStats.ReceiverBytesReceived	808	Number of bytes of data received by receiver channels from network attached WebSphere MQ Queue Managers.	All WebSphere MQ Queue Managers that are or have been connected to this application server	CountStatistic	Low
ReceiverBytesSentCount	MQLinkStats.ReceiverBytesSent	807	Number of bytes of data sent by receiver channels to network attached WebSphere MQ Queue Managers.	All WebSphere MQ Queue Managers that are or have been connected to this application server	CountStatistic	Low
SenderBytesReceivedCount	MQLinkStats.SenderBytesReceived	806	Number of bytes of data received by sender channels from network attached WebSphere MQ Queue Managers.	All WebSphere MQ Queue Managers that are or have been connected to this application server	CountStatistic	Low
SenderBytesSentCount	MQLinkStats.SenderBytesSent	805	Number of bytes of data sent by sender channels to network attached WebSphere MQ Queue Managers.	All WebSphere MQ Queue Managers that are or have been connected to this application server	CountStatistic	Low
ShortRetriesCount	MQLinkStats.ShortRetries	809	Number of short retries. This indicates the number of times channels were disconnected and could not be re-established for short periods of time.	All WebSphere MQ Queue Managers that are or have been connected to this application server	CountStatistic	Low
WritesBlockedCount	MQLinkStats.WritesBlocked	813	Number of write operations that could not be completed immediately. This number can be used as an indicator of network congestion when communicating with WebSphere MQ Queue Managers.	All WebSphere MQ Queue Managers that are or have been connected to this application server	CountStatistic	Low

resource_name → **SIB Service** → **SIB Communications** → **WMQ Client Links** → **Standard Statistics**

Name	Key	ID	Description	Granularity	Type	Level
APICallsServicedCount	MQClientLinkStats.APICallsServiced	906	The number of MQ API call requests serviced on behalf of WebSphere MQ JMS clients.	All WebSphere MQ JMS clients that are or have been connected to this application server	CountStatistic	Low
BatchesSentCount	MQClientLinkStats.BatchesSent	901	Number of batches of messages sent to network attached MQJMS clients.	All WebSphere MQ JMS clients that are or have been connected to this application server	CountStatistic	Low
BytesReceivedCount	MQClientLinkStats.BytesReceived	905	Number of bytes of data received from network attached WebSphere MQ JMS clients. This includes bytes of message data as well as bytes of data used to control the flow of messages.	All WebSphere MQ JMS clients that are or have been connected to this application server	CountStatistic	Low
BytesSentCount	MQClientLinkStats.BytesSent	904	Number of bytes of data sent to network attached WebSphere MQ JMS clients. This includes bytes of message data as well as bytes of data used to control the flow of messages.	All WebSphere MQ JMS clients that are or have been connected to this application server	CountStatistic	Low
ClientsAttachedCount	MQClientLinkStats.ClientsAttached	908	The current number of WebSphere MQ JMS clients attached to this application server.	WebSphere MQ JMS clients that are currently attached.	CountStatistic	Low
CommsErrorsCount	MQClientLinkStats.CommsErrors	907	The number of errors that have cause connections to WebSphere MQ JMS clients to be dropped.	All WebSphere MQ JMS clients that are or have been connected to this application server	CountStatistic	Low
MessagesReceivedCount	MQClientLinkStats.MessagesReceived	903	Number of messages received from network attached WebSphere MQ JMS clients.	All WebSphere MQ JMS clients that are or have been connected to this application server	CountStatistic	Low
MessagesSentCount	MQClientLinkStats.MessagesSent	902	Number of messages sent to network attached WebSphere MQ JMS clients.	All WebSphere MQ JMS clients that are or have been connected to this application server	CountStatistic	Low

resource_name → **SIB Service** → **SIB Communications** → **WMQ Client Links** → **Standard Statistics**

ReadsBlockedCount	MQClientLinkStats.ReadsBlocked	910	Number of read operations that could not be completed immediately. This number can be used as an indicator of network congestion when communicating with WebSphere MQ JMS clients.	All WebSphere MQ JMS clients that are or have been connected to this application server	CountStatistic	Low
WritesBlockedCount	MQClientLinkStats.WritesBlocked	909	Number of write operations that could not be completed immediately. This number can be used as an indicator of network congestion when communicating with WebSphere MQ JMS clients.	All WebSphere MQ JMS clients that are or have been connected to this application server	CountStatistic	Low

Web services gateway counters:

These service integration bus Web services gateway counters are part of the performance monitoring infrastructure (PMI), which provides server-side monitoring and a client-side API to retrieve performance information. Examples of Web services gateway counters are the number of synchronous and asynchronous requests and responses.

Counter definitions*:* To see lists of available PMI counters, use the administrative console to navigate to **Monitoring and Tuning** → **Performance Monitoring Infrastructure (PMI)** → ***resource_name*** → **[General Properties] Custom**.

resource_name → Web services gateway

Name	Key	ID	Description	Granularity	Type	Level	Overhead
AsynchronousRequestCount	wsgwModule.asynchronousRequests	3	The number of asynchronous requests made.	Per Web service	CountStatistic	All	Low
AsynchronousResponseCount	wsgwModule.asynchronousResponses	4	The number of asynchronous responses made.	Per Web service	CountStatistic	All	Low
SynchronousRequestCount	wsgwModule.synchronousRequests	1	The number of synchronous requests made.	Per Web service	CountStatistic	All	Low
SynchronousResponseCount	wsgwModule.synchronousResponses	2	The number of synchronous responses made.	Per Web service	CountStatistic	All	Low

SIP PMI counters

The Session Initiation Protocol (SIP) provides the following counters in the WebSphere Performance Monitoring Infrastructure (PMI) to monitor the performance of SIP.

Counter definitions: **Sip container module**

Name	Key	Description	Granularity	Type	Level	Overhead	ID
Incoming traffic	incoming.traffic	Average number of messages handled by the container calculated over configurable period	Per server	CountStatistic	All	Low	3
New SIP App sessions	new.sip.app.session	Average number of new SIP application sessions created in the container and calculated over configurable period	Per server	CountStatistic	All	Low	4
Response Time	response.time	The average amount of time that it takes between when a message gets into the container and when a response is sent from the container.	Per server	CountStatistic	All	Low	5
Queue Size	queue.size	Size of the invoke queue in WebSphere Application Server	Per server	CountStatistic	All	Low	6

Session module

Name	Key	Description	Granularity	Type	Level	Overhead	ID
Number of active SIP sessions	active.sip.sessions	The number of SIP sessions belongs to each application	Per application	CountStatistic	All	Low	11
Number of active SIP application sessions	active.sip.app.sessions	The number of SIP application sessions belongs to each application	Per application	CountStatistic	All	Low	12

Inbound request module

Name	Key	Description	Granularity	Type	Level	Overhead	ID
Number of Inbound NOT SIP STANDARD requests	inbound.request.other	The number of inbound NOT SIP STANDARD requests that belong to each application	Per application	CountStatistic	All	Low	60
Number of Inbound REGISTER requests	inbound.request.register	The number of inbound REGISTER requests that belong to each application	Per application	CountStatistic	All	Low	61
Number of Inbound INVITE requests	inbound.request.invite	The number of inbound INVITE requests that belong to each application	Per application	CountStatistic	All	Low	62
Number of Inbound ACK requests	inbound.request.ack	The number of Inbound ACK requests that belong to each application	Per application	CountStatistic	All	Low	63
Number of Inbound OPTIONS requests	inbound.request.options	The number of Inbound OPTIONS requests that belong to each application	Per application	CountStatistic	All	Low	64
Number of Inbound BYE requests	inbound.request.bye	The number of Inbound BYE requests that belong to each application	Per application	CountStatistic	All	Low	65
Number of Inbound CANCEL requests	inbound.request.cancel	The number of Inbound CANCEL requests that belong to each application	Per application	CountStatistic	All	Low	66
Number of Inbound PRACK requests	inbound.request.prack	The number of Inbound PRACK requests that belong to each application	Per application	CountStatistic	All	Low	67
Number of Inbound INFO requests	inbound.request.info	The number of Inbound INFO requests that belong to each application	Per application	CountStatistic	All	Low	68
Number of Inbound SUBSCRIBE requests	inbound.request.subscribe	The number of Inbound SUBSCRIBE requests that belong to each application	Per application	CountStatistic	All	Low	69
Number of Inbound NOTIFY requests	inbound.request.notify	The number of Inbound NOTIFY requests that belong to each application	Per application	CountStatistic	All	Low	70
Number of Inbound MESSAGE requests	inbound.request.message	The number of Inbound MESSAGE requests that belong to each application	Per application	CountStatistic	All	Low	71
Number of Inbound PUBLISH requests	inbound.request.publish	The number of Inbound PUBLISH requests that belong to each application	Per application	CountStatistic	All	Low	72
Number of Inbound REFER requests	inbound.request.refer	The number of Inbound REFER requests that belong to each application	Per application	CountStatistic	All	Low	73
Number of Inbound UPDATE requests	inbound.request.update	The number of Inbound UPDATE requests that belong to each application	Per application	CountStatistic	All	Low	74

Inbound info response module

Name	Key	Description	Granularity	Type	Level	Overhead	ID
Number of Inbound 100 responses	inbound.response.info.100	The number of Inbound 100 (Trying) responses that belong to each application	Per application	CountStatistic	All	Low	1100
Number of Inbound 180 responses	inbound.response.info.180	The number of Inbound 180 (Ringing) responses that belong to each application	Per application	CountStatistic	All	Low	1180
Number of Inbound 181 responses	inbound.response.info.181	The number of Inbound 181 (Call being forwarded) responses that belong to each application	Per application	CountStatistic	All	Low	1181
Number of Inbound 182 responses	inbound.response.info.182	The number of Inbound 182 (Call Queued) responses that belong to each application	Per application	CountStatistic	All	Low	1182
Number of Inbound 183 responses	inbound.response.info.183	The number of Inbound 183 (Session Progress) responses that belong to each application	Per application	CountStatistic	All	Low	1183

Inbound success response module

Name	Key	Description	Granularity	Type	Level	Overhead	ID
Number of successful Inbound 200 responses	inbound.response.success.200	The number of Inbound 200 (OK) responses that belong to each application	Per application	CountStatistic	All	Low	1200
Number of successful Inbound 202 responses	inbound.response.success.202	The number of Inbound 202 (Accepted) responses that belong to each application	Per application	CountStatistic	All	Low	1202

Inbound redirect responses module

Name	Key	Description	Granularity	Type	Level	Overhead	ID
Number of Inbound 300 responses	inbound.response.redirect.300	The number of Inbound 300 (Multiple choices) responses that belong to each application	Per application	CountStatistic	All	Low	1300
Number of Inbound 301 responses	inbound.response.redirect.301	The number of Inbound 301 (Moved Permanently) responses that belong to each application	Per application	CountStatistic	All	Low	1301
Number of Inbound 302 responses	inbound.response.redirect.302	The number of Inbound 302 (Moved Temporarily) responses that belong to each application	Per application	CountStatistic	All	Low	1302
Number of Inbound 305 responses	inbound.response.redirect.305	The number of Inbound 305 (Use Proxy) responses that belong to each application	Per application	CountStatistic	All	Low	1305
Number of Inbound 380 responses	inbound.response.redirect.380	The number of Inbound 380 (Alternative Service) responses that belong to each application	Per application	CountStatistic	All	Low	1380

Inbound fail response module

Name	Key	Description	Granularity	Type	Level	Overhead	ID
Number of Inbound 400 responses	inbound.response.fail.400	The number of Inbound 400 (Bad Request) responses that belong to each application	Per application	CountStatistic	All	Low	1400
Number of Inbound 401 responses	inbound.response.fail.401	The number of Inbound 401 (Unauthorized) responses that belong to each application	Per application	CountStatistic	All	Low	1401
Number of Inbound 402 responses	inbound.response.fail.402	The number of Inbound 402 (Payment Required) responses that belong to each application	Per application	CountStatistic	All	Low	1402
Number of Inbound 403 responses	inbound.response.fail.403	The number of Inbound 403 (Forbidden) responses that belong to each application	Per application	CountStatistic	All	Low	1403
Number of Inbound 404 responses	inbound.response.fail.404	The number of Inbound 404 (Not Found) responses that belong to each application	Per application	CountStatistic	All	Low	1404
Number of Inbound 405 responses	inbound.response.fail.405	The number of Inbound 405 (Method Not Allowed) responses that belong to each application	Per application	CountStatistic	All	Low	1405
Number of Inbound 406 responses	inbound.response.fail.406	The number of Inbound 406 (Not Acceptable) responses that belong to each application	Per application	CountStatistic	All	Low	1406
Number of Inbound 407 responses	inbound.response.fail.407	The number of Inbound 407 (Proxy Authentication Required) responses that belong to each application	Per application	CountStatistic	All	Low	1407
Number of Inbound 408 responses	inbound.response.fail.408	The number of Inbound 408 (Request Timeout) responses that belong to each application	Per application	CountStatistic	All	Low	1408
Number of Inbound 410 responses	inbound.response.fail.410	The number of Inbound 410 (Gone) responses that belong to each application	Per application	CountStatistic	All	Low	1410
Number of Inbound 413 responses	inbound.response.fail.413	The number of Inbound 413 (Request Entity Too Large) responses that belong to each application	Per application	CountStatistic	All	Low	1413
Number of Inbound 414 responses	inbound.response.fail.414	The number of Inbound 414 (Request URI Too Long) responses that belong to each application	Per application	CountStatistic	All	Low	1414

Number of Inbound 415 responses	inbound.response.fail.415	The number of Inbound 415 (Unsupported Media Type) responses that belong to each application	Per application	CountStatistic	All	Low	1415
Number of Inbound 416 responses	inbound.response.fail.416	The number of Inbound 416 (Unsupported URI Scheme) responses that belong to each application	Per application	CountStatistic	All	Low	1416
Number of Inbound 420 responses	inbound.response.fail.420	The number of Inbound 420 (Bad Extention) responses that belong to each application	Per application	CountStatistic	All	Low	1420
Number of Inbound 421 responses	inbound.response.fail.421	The number of Inbound 421 (Extention Required) responses that belong to each application	Per application	CountStatistic	All	Low	1421
Number of Inbound 423 responses	inbound.response.fail.423	The number of Inbound 423 (Interval Too Brief) responses that belong to each application	Per application	CountStatistic	All	Low	1423
Number of Inbound 480 responses	inbound.response.fail.480	The number of Inbound 480 (Temporarly Unavailable) responses that belong to each application	Per application	CountStatistic	All	Low	1480
Number of Inbound 481 responses	inbound.response.fail.481	The number of Inbound 481 (Call Leg Done) responses that belong to each application	Per application	CountStatistic	All	Low	1481
Number of Inbound 482 responses	inbound.response.fail.482	The number of Inbound 482 (Loop Detected) responses that belong to each application	Per application	CountStatistic	All	Low	1482
Number of Inbound 483 responses	inbound.response.fail.483	The number of Inbound 483 (Too Many Hops) responses that belong to each application	Per application	CountStatistic	All	Low	1483
Number of Inbound 484 responses	inbound.response.fail.484	The number of Inbound 484 (Address Incomplete) responses that belong to each application	Per application	CountStatistic	All	Low	1484
Number of Inbound 485 responses	inbound.response.fail.485	The number of Inbound 485 (Ambiguous) responses that belong to each application	Per application	CountStatistic	All	Low	1485
Number of Inbound 486 responses	inbound.response.fail.486	The number of Inbound 486 (Busy Here) responses that belong to each application	Per application	CountStatistic	All	Low	1486

Number of Inbound 487 responses	inbound.response.fail.487	The number of Inbound 487 (Request Terminated) responses that belong to each application	Per application	CountStatistic	All	Low	1487
Number of Inbound 488 responses	inbound.response.fail.488	The number of Inbound 488 (Not Acceptable Here) responses that belong to each application	Per application	CountStatistic	All	Low	1488
Number of Inbound 491 responses	inbound.response.fail.491	The number of Inbound 491 (Request Pending) responses that belong to each application	Per application	CountStatistic	All	Low	1491
Number of Inbound 493 responses	inbound.response.fail.493	The number of Inbound 493 (Undecipherable) responses that belong to each application	Per application	CountStatistic	All	Low	1493

Inbound server fail response module

Name	Key	Description	Granularity	Type	Level	Overhead	ID
Number of Inbound 500 responses	inbound.response.serverFail.500	The number of Inbound 500 (Server Internal Error) responses that belong to each application	Per application	CountStatistic	All	Low	1500
Number of Inbound 501 responses	inbound.response.serverFail.501	The number of Inbound 501 (Not Implemented) responses that belong to each application	Per application	CountStatistic	All	Low	1501
Number of Inbound 502 responses	inbound.response.serverFail.502	The number of Inbound 502 (Bad Gateway) responses that belong to each application	Per application	CountStatistic	All	Low	1502
Number of Inbound 503 responses	inbound.response.serverFail.503	The number of Inbound 503 (Service Unavailable) responses that belong to each application	Per application	CountStatistic	All	Low	1503
Number of Inbound 504 responses	inbound.response.serverFail.504	The number of Inbound 504 (Server Timeout) responses that belong to each application	Per application	CountStatistic	All	Low	1504
Number of Inbound 505 responses	inbound.response.serverFail.505	The number of Inbound 505 (Version Not Supported) responses that belong to each application	Per application	CountStatistic	All	Low	1505
Number of Inbound 513 responses	inbound.response.serverFail.513	The number of Inbound 513 (Message Too Large) responses that belong to each application	Per application	CountStatistic	All	Low	1513

Inbound global fail response module

Name	Key	Description	Granularity	Type	Level	Overhead	ID
Number of Inbound 600 responses	inbound.response.globalFail.600	The number of Inbound 600 (Busy Everywhere) responses that belong to each application	Per application	CountStatistic	All	Low	1600
Number of Inbound 603 responses	inbound.response.globalFail.603	The number of Inbound 603 (Decline) responses that belong to each application	Per application	CountStatistic	All	Low	1603
Number of Inbound 604 responses	inbound.response.globalFail.604	The number of Inbound 604 (Does Not Exit Anywhere) responses that belong to each application	Per application	CountStatistic	All	Low	1604
Number of Inbound 606 responses	inbound.response.globalFail.606	The number of Inbound 606 (Not Acceptable Anywhere) responses that belong to each application	Per application	CountStatistic	All	Low	1606

Outbound request module

Name	Key	Description	Granularity	Type	Level	Overhead	ID
Number of Outbound NOT SIP STANDARD requests	outbound.request.other	The number of Outbound NOT SIP STANDARD requests that belong to each application	Per application	CountStatistic	All	Low	80
Number of Outbound REGISTER requests	outbound.request.register	The number of Outbound REGISTER requests that belong to each application	Per application	CountStatistic	All	Low	81
Number of Outbound INVITE requests	outbound.request.invite	The number of Outbound INVITE requests that belong to each application	Per application	CountStatistic	All	Low	82
Number of Outbound ACK requests	outbound.request.ack	The number of Outbound ACK requests that belong to each application	Per application	CountStatistic	All	Low	83
Number of Outbound OPTIONS requests	outbound.request.options	The number of Outbound OPTIONS requests that belong to each application	Per application	CountStatistic	All	Low	84
Number of Outbound BYE requests	outbound.request.bye	The number of Outbound BYE requests that belong to each application	Per application	CountStatistic	All	Low	85
Number of Outbound CANCEL requests	outbound.request.cancel	The number of Outbound CANCEL requests that belong to each application	Per application	CountStatistic	All	Low	86
Number of Outbound PRACK requests	outbound.request.prack	The number of Outbound PRACK requests that belong to each application	Per application	CountStatistic	All	Low	87
Number of Outbound INFO requests	outbound.request.info	The number of Outbound INFO requests that belong to each application	Per application	CountStatistic	All	Low	88
Number of Outbound SUBSCRIBE requests	outbound.request.subscribe	The number of Outbound SUBSCRIBE requests that belong to each application	Per application	CountStatistic	All	Low	89
Number of Outbound NOTIFY requests	outbound.request.notify	The number of Outbound NOTIFY requests that belong to each application	Per application	CountStatistic	All	Low	90
Number of Outbound MESSAGE requests	outbound.request.message	The number of Outbound MESSAGE requests that belong to each application	Per application	CountStatistic	All	Low	91
Number of Outbound PUBLISH requests	outbound.request.publish	The number of Outbound PUBLISH requests that belong to each application	Per application	CountStatistic	All	Low	92
Number of Outbound REFER requests	outbound.request.refer	The number of Outbound REFER requests that belong to each application	Per application	CountStatistic	All	Low	93
Number of Outbound UPDATE requests	outbound.request.update	The number of Outbound UPDATE requests that belong to each application	Per application	CountStatistic	All	Low	94

Outbound info response module

Name	Key	Description	Granularity	Type	Level	Overhead	ID
Number of Outbound 100 responses	outbound.response.info.100	The number of Outbound 100 (Trying) responses that belong to each application	Per application	CountStatistic	All	Low	2100
Number of Outbound 180 responses	outbound.response.info.180	The number of Outbound 180 (Ringing) responses that belong to each application	Per application	CountStatistic	All	Low	2180
Number of Outbound 181 responses	outbound.response.info.181	The number of Outbound 181 (Call being forwarded) responses that belong to each application	Per application	CountStatistic	All	Low	2181
Number of Outbound 182 responses	outbound.response.info.182	The number of Outbound 182 (Call Queued) responses that belong to each application	Per application	CountStatistic	All	Low	2182
Number of Outbound 183 responses	outbound.response.info.183	The number of Outbound 183 (Session Progress) responses that belong to each application	Per application	CountStatistic	All	Low	2183

Outbound success response module

Name	Key	Description	Granularity	Type	Level	Overhead	ID
Number of Outbound 200 responses	outbound.response.success.200	The number of Outbound 200 (OK) responses that belong to each application	Per application	CountStatistic	All	Low	2200
Number of Outbound 202 responses	outbound.response.success.202	The number of Outbound 202 (Accepted) responses that belong to each application	Per application	CountStatistic	All	Low	2202

Outbound redirect response module

Name	Key	Description	Granularity	Type	Level	Overhead	ID
Number of Outbound 300 responses	outbound.response.redirect.300	The number of Outbound 300 (Multiple choices) responses that belong to each application	Per application	CountStatistic	All	Low	2300
Number of Outbound 301 responses	outbound.response.redirect.301	The number of Outbound 301 (Moved Permanently) responses that belong to each application	Per application	CountStatistic	All	Low	2301
Number of Outbound 302 responses	outbound.response.redirect.302	The number of Outbound 302 (Moved Temporarily) responses that belong to each application	Per application	CountStatistic	All	Low	2302
Number of Outbound 305 responses	outbound.response.redirect.305	The number of Outbound 305 (Use Proxy) responses that belong to each application	Per application	CountStatistic	All	Low	2305
Number of Outbound 380 responses	outbound.response.redirect.380	The number of Outbound 380 (Alternative Service) responses that belong to each application	Per application	CountStatistic	All	Low	2380

Outbound fail response module

Name	Key	Description	Granularity	Type	Level	Overhead	ID
Number of Outbound 400 responses	outbound.response.fail.400	The number of Outbound 400 (Bad Request) responses that belong to each application	Per application	CountStatistic	All	Low	2400
Number of Outbound 401 responses	outbound.response.fail.401	The number of Outbound 401 (Unauthorized) responses that belong to each application	Per application	CountStatistic	All	Low	2401
Number of Outbound 402 responses	outbound.response.fail.402	The number of Outbound 402 (Payment Required) responses that belong to each application	Per application	CountStatistic	All	Low	2402
Number of Outbound 403 responses	outbound.response.fail.403	The number of Outbound 403 (Forbidden) responses that belong to each application	Per application	CountStatistic	All	Low	2403
Number of Outbound 404 responses	outbound.response.fail.404	The number of Outbound 404 (Not Found) responses that belong to each application	Per application	CountStatistic	All	Low	2404
Number of Outbound 405 responses	outbound.response.fail.405	The number of Outbound 405 (Method Not Allowed) responses that belong to each application	Per application	CountStatistic	All	Low	2405
Number of Outbound 406 responses	outbound.response.fail.406	The number of Outbound 406 (Not Acceptable) responses that belong to each application	Per application	CountStatistic	All	Low	2406
Number of Outbound 407 responses	outbound.response.fail.407	The number of Outbound 407 (Proxy Authentication Required) responses that belong to each application	Per application	CountStatistic	All	Low	2407
Number of Outbound 408 responses	outbound.response.fail.408	The number of Outbound 408 (Request Timeout) responses that belong to each application	Per application	CountStatistic	All	Low	2408
Number of Outbound 410 responses	outbound.response.fail.410	The number of Outbound 410 (Gone) responses that belong to each application	Per application	CountStatistic	All	Low	2410
Number of Outbound 413 responses	outbound.response.fail.413	The number of Outbound 413 (Request Entity Too Large) responses that belong to each application	Per application	CountStatistic	All	Low	2413
Number of Outbound 414 responses	outbound.response.fail.414	The number of Outbound 414 (Request URI Too Long) responses that belong to each application	Per application	CountStatistic	All	Low	2414

Number of Outbound 415 responses	outbound.response.fail.415	The number of Outbound 415 (Unsupported Media Type) responses that belong to each application	Per application	CountStatistic	All	Low	2415
Number of Outbound 416 responses	outbound.response.fail.416	The number of Outbound 416 (Unsupported URI Scheme) responses that belong to each application	Per application	CountStatistic	All	Low	2416
Number of Outbound 420 responses	outbound.response.fail.420	The number of Outbound 420 (Bad Extention) responses that belong to each application	Per application	CountStatistic	All	Low	2420
Number of Outbound 421 responses	outbound.response.fail.421	The number of Outbound 421 (Extention Required) responses that belong to each application	Per application	CountStatistic	All	Low	2421
Number of Outbound 423 responses	outbound.response.fail.423	The number of Outbound 423 (Interval Too Brief) responses that belong to each application	Per application	CountStatistic	All	Low	2423
Number of Outbound 480 responses	outbound.response.fail.480	The number of Outbound 480 (Temporarly Unavailable) responses that belong to each application	Per application	CountStatistic	All	Low	2480
Number of Outbound 481 responses	outbound.response.fail.481	The number of Outbound 481 (Call Leg Done) responses that belong to each application	Per application	CountStatistic	All	Low	2481
Number of Outbound 482 responses	outbound.response.fail.482	The number of Outbound 482 (Loop Detected) responses that belong to each application	Per application	CountStatistic	All	Low	2482
Number of Outbound 483 responses	outbound.response.fail.483	The number of Outbound 483 (Too Many Hops) responses that belong to each application	Per application	CountStatistic	All	Low	2483
Number of Outbound 484 responses	outbound.response.fail.484	The number of Outbound 484 (Address Incomplete) responses that belong to each application	Per application	CountStatistic	All	Low	2484
Number of Outbound 485 responses	outbound.response.fail.485	The number of Outbound 485 (Ambiguous) responses that belong to each application	Per application	CountStatistic	All	Low	2485
Number of Outbound 486 responses	outbound.response.fail.486	The number of Outbound 486 (Busy Here) responses that belong to each application	Per application	CountStatistic	All	Low	2486

Number of Outbound 487 responses	outbound.response.fail.487	The number of Outbound 487 (Request Terminated) responses that belong to each application	Per application	CountStatistic	All	Low	2487
Number of Outbound 488 responses	outbound.response.fail.488	The number of Outbound 488 (Not Acceptable Here) responses that belong to each application	Per application	CountStatistic	All	Low	2488
Number of Outbound 491 responses	outbound.response.fail.491	The number of Outbound 491 (Request Pending) responses that belong to each application	Per application	CountStatistic	All	Low	2491
Number of Outbound 493 responses	outbound.response.fail.493	The number of Outbound 493 (Undecipherable) responses that belong to each application	Per application	CountStatistic	All	Low	2493

Outbound server fail response module

Name	Key	Description	Granularity	Type	Level	Overhead	ID
Number of Outbound 500 responses	outbound.response.serverFail.500	The number of Outbound 500 (Server Internal Error) responses that belong to each application	Per application	CountStatistic	All	Low	2500
Number of Outbound 501 responses	outbound.response.serverFail.501	The number of Outbound 501 (Not Implemented) responses that belong to each application	Per application	CountStatistic	All	Low	2501
Number of Outbound 502 responses	outbound.response.serverFail.502	The number of Outbound 502 (Bad Gateway) responses that belong to each application	Per application	CountStatistic	All	Low	2502
Number of Outbound 503 responses	outbound.response.serverFail.503	The number of Outbound 503 (Service Unavailable) responses that belong to each application	Per application	CountStatistic	All	Low	2503
Number of Outbound 504 responses	outbound.response.serverFail.504	The number of Outbound 504 (Server Timeout) responses that belong to each application	Per application	CountStatistic	All	Low	2504
Number of Outbound 505 responses	outbound.response.serverFail.505	The number of Outbound 505 (Version Not Supported) responses that belong to each application	Per application	CountStatistic	All	Low	2505
Number of Outbound 513 responses	outbound.response.serverFail.513	The number of Outbound 513 (Message Too Large) responses that belong to each application	Per application	CountStatistic	All	Low	2513

Outbound global fail response module

Name	Key	Description	Granularity	Type	Level	Overhead	ID
Number of Outbound 600 responses	outbound.response.globalFail.600	The number of Outbound 600 (Busy Everywhere) responses that belong to each application	Per application	CountStatistic	All	Low	2600
Number of Outbound 603 responses	outbound.response.globalFail.603	The number of Outbound 603 (Decline) responses that belong to each application	Per application	CountStatistic	All	Low	2603
Number of Outbound 604 responses	outbound.response.globalFail.604	The number of Outbound 604 (Does Not Exit Anywhere) responses that belong to each application	Per application	CountStatistic	All	Low	2604
Number of Outbound 606 responses	outbound.response.globalFail.606	The number of Outbound 606 (Not Acceptable Anywhere) responses that belong to each application	Per application	CountStatistic	All	Low	2606

PMI data collection

PMI data collection can occur in three different interfaces, Java Management Extension (JMX) interface (J2EE MBeans and WebSphere Application Server Perf MBean), Performance Servlet, or PMI client API (deprecated).

JMX Interface

JMX interface is part of the J2EE specification and the recommended way to gather WebSphere Application Server performance data. PMI data can be gathered from the J2EE managed object MBeans or the WebSphere Application Server PMI Perf MBean. While the J2EE MBeans provide performance data about the specific component, the Perf MBean acts as a gateway to the WebSphere Application Server PMI service, and provides access to the performance data for all the components.

Performance Servlet

Performance Servlet provides a way to use an HTTP request to query the PMI data for the entire WebSphere Application Server administrative domain. Since the servlet provides the performance data through HTTP, issues such as firewalls are trivial to resolve. The performance servlet outputs the PMI data as an XML document.

PMI client API (deprecated)

PMI client API provides a wrapper class to deliver PMI data to a client. The PMI client API uses the JMX infrastructure and the Perf MBean to retrieve the PMI data. PMI client provides the data using a WebSphere Application Server-specific data structure.

PMI client API provides a wrapper class to deliver PMI data to a client. The PMI client API uses the JMX infrastructure and the Perf MBean to retrieve the PMI data. PMI client provides the data using a WebSphere Application Server-specific data structure.

Custom PMI API

You can create specific statistics to best meet your monitoring interests by using custom PMI API.

PMI can be extended using the Custom PMI API to create application specific statistics. For example, a stock trading application can use Custom PMI API to create business specific statistics like ″number of stock sell transactions″ and ″number of stock buy transactions″.

Note that PMI provides detailed performance data about various runtime and application components. Starting with WebSphere Application Server Version 6.0, PMI offers approximately 180 or more performance statistics. Before creating new statistics, it is important to make sure that the same data is not captured by PMI already.

With WebSphere PMI, application developers can add their own application-specific instrumentation. The Custom PMI API simplifies the process of ″PMI enabling″ an application by providing an easy to use API. The statistics created via the Custom PMI can be accessed via the standard PMI and JMX interfaces that are used by monitoring tools including the Tivoli Performance Viewer.

PMI instrumentation is based on the Java Platform, Enterprise Edition (Java EE) 1.4 standard. As a result, Custom PMI supports all the Statistic types (CountStatistic, TimeStatistic, RangeStatistic, and BoundedRangeStatistic) defined in the JSR-77 Performance Data Framework. Custom PMI does not support user-defined Statistic types.

What you need to know

PMI collects performance data on runtime applications and provides interfaces that allow external applications to monitor the performance data.

With server side PMI, application developers can add their own instrumentation to their applications to help monitor their own predefined performance metrics.

Key features of Custom PMI:

- Create a custom Stats or PMI (Stats is Java EE terminology) module using an XML template.
- Used by the application to instrument code.
- Statistics in the custom Stats module can be accessed via the standard PMI and JMX interfaces that are used by monitoring tools, including the Tivoli Performance Viewer.
- PMI instrumentation is based on the Java EE 1.4 standard. As a result, Custom PMI supports all the Statistic types (CountStatistic, TimeStatistic, RangeStatistic, and BoundedRangeStatistic) defined in the JSR-77 Performance Data Framework.
- Custom PMI does not support user-defined Statistic types.

PMI is for application server performance monitoring, and the data collected by PMI is used to tune the application server resources such as pools, queues, and caches, etc. Since performance instrumentation and statistics can have considerable impact on the application server performance, it is necessary that every statistic added via Custom PMI is relevant towards solving a performance problem. When designing statistics, consider the following issues:

- Significance of the statistic with respect to solving performance problems.
- Relevance to tuning or configuring the application.
- Avoiding data redundancy and unnecessary frequent data updates.

Enabling PMI data collection

Enable PMI data collection to diagnose problems and tune application performance.

Before you begin

Performance Monitoring Infrastructure (PMI) must be enabled (by default PMI is enabled) before collecting any performance data. PMI should be enabled before the server starts. If PMI is enabled after the server is started, the server needs to be restarted to start the PMI.

About this task

When PMI service is enabled, the monitoring of individual components can be enabled or disabled dynamically. PMI provides four predefined statistic sets that can be used to enable a set of statistics. The following table provides details about the statistic sets. If the predefined statistic sets does not meet your monitoring requirement, the **Custom** option can be used to selectively enable or disable individual statistics.

Statistic set	Description
None	All statistics are disabled.
Basic	Statistics specified in J2EE 1.4, as well as top statistics like CPU usage and live HTTP sessions are enabled. This set is enabled *out-of-the-box* and provides basic performance data about runtime and application components.
Extended	Basic set plus key statistics from various WebSphere Application Server components like WLM and Dynamic caching are enabled. This set provides detailed performance data about various runtime and application components.
All	All statistics are enabled.
Custom	Enable or disable statistics selectively.

Custom setting

WebSphere Application Server Version 6.0 introduces fine-grained control to enable/disable statistics individually. The fine-grained control is available under the custom statistic set.

Though the Version 5.x monitoring levels {N, L, M, H, X} are deprecated in Version 6.0 and above, the 5.x PMI APIs are supported for backward compatibility. It is possible to use a Version 6.0 or later API to set the monitoring level and a Version 5.0 API to get the monitoring level. A new level 'F' – "fine-grained" is introduced to indicate to the Version 5.x API that the fine-grained or Version 6.x monitoring specification is in effect. This new level 'F' will be returned if a V5.x API is used to get the monitoring level from a server using Version 6.x monitoring specification.

In WebSphere Application Server Version 4.0 and Version 5.0, the statistics were enabled based on a monitoring/instrumentation level. The levels are None, Low, Medium, High and Max {N, L, M, H, X}. Enabling at a given level will include all the statistics at the given level plus counters from levels below the given level. So if you enable the Web Application module at level Medium {M} it enables all the counters at level M, plus all the Low {L} level counters. See "PMI data collection" on page 173 for more information.

Sequential Update

In order to minimize the monitoring overhead, the updates to CountStatistic, AverageStatistic, and TimeStatistic are not synchronized. Since these statistic types track total and average, the extra accuracy is generally not worth the performance cost. The RangeStatistic and BoundedRangeStatistic are very sensitive; therefore, they are always are synchronized. If needed, updates to all the statistic types can be synchronized by checking the **Use sequential update** check box.

Enable PMI using one of the following methods:

- "Enabling PMI using the administrative console"
- "Enabling PMI using the wsadmin tool" on page 178
- "Enabling the Java virtual machine profiler data" on page 186

Enabling PMI using the administrative console

When PMI service is enabled, the monitoring of individual components can be enabled or disabled dynamically.

About this task

To monitor performance data through the Performance Monitoring Infrastructure (PMI), you must first enable PMI through the administrative console.

1. Open the administrative console.
2. Click **Servers** > **Application Servers** in the console navigation tree.
3. Click a *server*.
4. Click the **Configuration** tab.
5. Click **Performance Monitoring Infrastructure (PMI)** under Performance.
6. Select the **Enable Performance Monitoring Infrastructure (PMI)** check box.
7. Optionally, select the check box **Use sequential counter updates** to enable precise statistic update.
8. Optionally, choose a statistic set that needs to be monitored under **Currently Monitored Statistic Set**.
9. Optionally, click on **Custom** to selectively enable or disable statistics. Choose a component from the left side tree and enable or disable statistics on the right side table. Go back to the main PMI configuration page by clicking the **Performance Monitoring Infrastructure** link.
10. Click **Apply** or **OK**.
11. Click **Save**.

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

What to do next

When in the **Configuration** tab, settings apply after the server is restarted.

Performance Monitoring Infrastructure settings

Use this page to specify settings for performance monitoring, including enabling performance monitoring, selecting the Performance Monitoring Infrastructure (PMI) module and setting monitoring levels.

To view this administrative console page, click **Servers** > **Application Servers** > *server* > **Performance Monitoring Infrastructure (PMI)**.

Enable Performance Monitoring Infrastructure (PMI):

Specifies whether the application server attempts to enable Performance Monitoring Infrastructure (PMI). If an application server is started when the PMI is disabled, you have to restart the server in order to enable it.

Use sequential counter updates:

Specifies whether access to PMI counters (for example, updates to the counters) is sequential. There is some performance overhead associated with enabling sequential update. The default setting is false.

Persist my changes:

Specifies whether changes to the runtime are also persisted for use on subsequent server startups, and not just the current server runtime. The default setting is false.

Selecting the Persist my changes option ensures that the counter settings are retained after you restart the server. This option itself does not remain selected the next time you come to the panel.

Currently monitored statistic set:

Specifies a pre-defined set of Performance Monitoring Infrastructure (PMI) statistics for all components in the server.

As part of fine-grained control feature, WebSphere Application Server provides new statistic sets, which are pre-defined, fixed server-side sets based on the PMI statistic usage scenarios. This enhancement allows a set of statistics to be enabled via a single action or call. The following statistic sets are defined:

None All statistics are disabled.

Basic Provides basic monitoring for application server resources and applications. This includes J2EE components, HTTP session information, CPU usage information, and the top 38 statistics. This is the default setting.

Extended
Provides extended monitoring, including the basic level of monitoring plus workload monitor, performance advisor, and Tivoli resource models. Extended provides key statistics from frequently used WebSphere Application Server components.

All Enables all statistics.

Custom
Provides fine-grained control with the ability to enable and disable individual statistics.

Custom monitoring level

Use this page to enable and disable specific PMI counters for individual PMI statistics.

To view this administrative console page, click **Servers** > **Application Servers** > *server* > **Performance Monitoring Infrastructure (PMI)** > **Custom**.

Custom monitoring level:

Click on the individual PMI statistic in the list on the left. The counters available for that module appear in the table on the right along with the counter type, a description of the counter, and its current status (Enabled or Disabled).

You can enable or disable each individual counter by selecting the counter and clicking the Enable or Disable button, respectively.

Custom monitoring level:

Click on the individual PMI module in the list on the left. The counters available for that statistic appear in the table on the right along with the counter type, a description of the counter, and its current status (Enabled or Disabled).

You can enable or disable each individual counter by selecting the counter and clicking the Enable or Disable button, respectively.

Note: Enabling or disabling a statistic set that contains subsets will affect the configuration of all the counters contained in the subsets. The parent statistic set that is selected passes down the current configuration set to its subsets, which can disable any counters that were individually selected within specific subsets. For best results, configure the parent statistic first, and then select specific counters within its subsets.

Performance Monitoring Infrastructure collection

Use this page to configure Performance Monitoring Infrastructure (PMI).

To view this administrative console page, click **Monitoring and Tuning** > **Performance Monitoring Infrastructure (PMI)** .

Name:

Specifies the logical name of the server.

The Name property is read only.

Node:

Specifies the name of the node holding the server.

The Node name property is read only.

Version:

Specifies the version of the WebSphere Application Server product on which the server runs.

The Version property is read only.

Enabling PMI using the wsadmin tool

Use this page to learn how to enable Performance Monitoring Infrastructure using command line instead of the administrative console.

About this task

You can use the command line to enable Performance Monitoring Infrastructure (PMI).

Run the **wsadmin** command, as described in Starting the wsadmin scripting client. Using **wsadmin**, you can invoke operations on Perf Mbean to obtain the PMI data, set or obtain PMI monitoring levels, and enable data counters.

Note: If PMI data are not enabled yet, you need to first enable PMI data by invoking setInstrumentationLevel operation on PerfMBean.

The following operations in Perf MBean can be used in **wsadmin**:

If an MBean is a StatisticProvider, and if you pass its ObjectName to getStatsObject, you get the Statistic data for that MBean. MBeans with the following MBean types are statistic providers:

- DynaCache
- EJBModule
- EntityBean
- JDBCProvider
- J2CResourceAdapter
- JVM
- MessageDrivenBean
- ORB
- Server
- SessionManager
- StatefulSessionBean
- StatelessSessionBean
- SystemMetrics
- ThreadPool
- TransactionService
- WebModule
- Servlet
- WLMAppServer
- WebServicesService
- WSGW

Example

Use the following sample Jacl commands with the wsadmin tool to obtain PMI data:

Obtain the Perf MBean ObjectName

```
wsadmin>set perfName [$AdminControl completeObjectName type=Perf,*]
wsadmin>set perfOName [$AdminControl makeObjectName $perfName]
```

Invoke getStatisticSet operation

Use this method to find the statistic set that is currently in effect:

```
Wsadmin> $AdminControl invoke $perfName getStatisticSet
```

This method returns one of the following values: basic, extended, all, none.

Invoke setStatisticSet operation

Use this method to enable monitoring using a statistic set.

The valid statistic set values are: basic, extended, all, none.

```
Wsadmin> set params [java::new {java.lang.Object[]} 1]
Wsadmin> $params set 0 [java::new java.lang.String extended]
Wsadmin> set sigs  [java::new {java.lang.String[]} 1]
Wsadmin> $sigs set 0 java.lang.String
Wsadmin> $AdminControl invoke_jmx $perfOName setStatisticSet
$params $sigs
```

Invoke getConfig operation

Use this method to find information about the statistics for a given component.

```
Wsadmin> set jvmName [$AdminControl completeObjectName type=JVM,*]

Wsadmin> set params [java::new {java.lang.Object[]} 1]
Wsadmin> $params set 0 [java::new javax.management.ObjectName $jvmName]
Wsadmin> set sigs  [java::new {java.lang.String[]} 1]
Wsadmin> $sigs set 0 javax.management.ObjectName

Wsadmin> $AdminControl invoke_jmx $perfOName getConfig $params
$sigs
```

This method returns the following:

```
    Stats type=jvmRuntimeModule, Description=The performance data from
the Java virtual machine run time.

    {name=UpTime, ID=4, type=CountStatistic, description=The amount of
time (in seconds) that the Java virtual machine has been running.,
unit=SECOND, level=low, statisticSet=basic, resettable=false,
aggregatable=true}

    {name=UsedMemory, ID=3, type=CountStatistic, description=The amount
of used memory (in KBytes) in the Java virtual machine run time.,
unit=KILOBYTE, level=low,
    statisticSet=basic, resettable=false, aggregatable=true}

    {name=FreeMemory, ID=2, type=CountStatistic, description=The free
memory (in KBytes) in the Java virtual machine run time.,
unit=KILOBYTE, level=low, statisticSet=all, resettable=false,
aggregatable=true}

    {name=HeapSize, ID=1, type=BoundedRangeStatistic, description=The
total memory (in KBytes) in the Java virtual machine run time.,
unit=KILOBYTE, level=high, statisticSet=basic, resettable=false,
aggregatable=true}
```

Invoke getCustomSetString operation

This operation provides the current monitoring specification in a string format:

```
Wsadmin> $AdminControl invoke $perfName getCustomSetString
```

The output looks similar to the following:

```
    jvmRuntimeModule=4,3,1:systemModule=2,1:threadPoolModule=4,3:thread
PoolModule>HAManager.thread.pool=4,3:threadPoolModule>MessageListenerTh
readPool=4,3:threadPoolModule>ORB.thread.pool=4,3:threadPoolModule>Serv
let.Engine.Transports=4,3:threadPoolModule>TCS_DEFAULT=4,3:transactionM
odule=4,19,16,18,3,7,6,1,14
```

This output indicates that statistic ID's 1, 3, and 4 are enabled in the JVM component. The description of the statistic IDs can be found using the above getConfig operation or using the API documentation. The

output contains the current monitoring specification for the entire server. The individual modules are separated by a **:**, and **>** is used as a separator within the module.

Invoke setCustomSetString operation

This operation can be used to enable or disable statistics selectively. In the following command the statistic IDs 1, 2, 3, and 4 are enabled for the JVM module. To enable all the statistic IDs use an asterisk (*).

```
    Wsadmin> set params [java::new {java.lang.Object[]} 2]
    Wsadmin> $params set 0 [java::new java.lang.String
jvmRuntimeModule=1,2,3,4]
    Wsadmin> $params set 1 [java::new java.lang.Boolean false]

    Wsadmin> set sigs  [java::new {java.lang.String[]} 2]
    Wsadmin> $sigs set 0 java.lang.String
    Wsadmin> $sigs set 1 java.lang.Boolean


    Wsadmin> $AdminControl invoke_jmx $perfOName setCustomSetString
$params $sigs
```

Invoke getStatsObject operation

This operation is used to get the statistics for a given MBean. The following example gets the statistics for the JVM:

```
    Wsadmin> set jvmName [$AdminControl completeObjectName type=JVM,*]
    Wsadmin> set params [java::new {java.lang.Object[]} 2]
    Wsadmin> $params set 0 [java::new javax.management.ObjectName
$jvmName]
    Wsadmin> $params set 1 [java::new java.lang.Boolean false]
    Wsadmin> set sigs  [java::new {java.lang.String[]} 2]
    Wsadmin> $sigs set 0 javax.management.ObjectName
    Wsadmin> $sigs set 1 java.lang.Boolean
    Wsadmin> $AdminControl invoke_jmx $perfOName getStatsObject $params
$sigs
    Stats name=jvmRuntimeModule, type=jvmRuntimeModule#
    {
    name=HeapSize, ID=1, description=The total memory (in KBytes) in
the Java virtual machine run time., unit=KILOBYTE, type=BoundedRangeStatistic, lowWaterMark=51200,
highWaterMark=263038, current=263038, integral=2.494158617766E12, lowerBound
=51200, upperBound=262144

    name=FreeMemory, ID=2, description=The free memory (in KBytes) in
the Java virtual machine run time., unit=KILOBYTE, type=CountStatistic,
count=53509

    name=UsedMemory, ID=3, description=The amount of used memory (in KBytes) in
the Java virtual machine run time., unit=KILOBYTE,
type=CountStatistic, count=209528

    name=UpTime, ID=4, description=The amount of time (in seconds) that
the Java virtual machine has been running., unit=SECOND,
type=CountStatistic, count=83050
}
```

Invoke getInstrumentationLevelString operation

Use *invoke*, because it has no parameter.

```
wsadmin>$AdminControl invoke $perfName
getInstrumentationLevelString
```

This command returns the following:

```
beanModule=H:cacheModule=H:connectionPoolModule=H:j2cModule=H:jvmRu
ntimeModule=H:orbPerfModule=H:servletSessionsModule=H:systemModule=
H:threadPoolModule=H:transactionModule=H:webAppModule=H
```

Note: You can change the level (n, l, m, h, x) in the above string and then pass it to setInstrumentationLevel method.

Invoke setInstrumentationLevel operation - enable/disable PMI counters

- sSet parameters (″pmi=l″ is the simple way to set all modules to the low level).

```
wsadmin>set params [java::new {java.lang.Object[]} 2]

wsadmin>$params set 0 [java::new java.lang.String pmi=l]
    wsadmin>$params set 1 [java::new java.lang.Boolean true]
```

- Set signatures.

```
wsadmin>set sigs  [java::new {java.lang.String[]} 2]

wsadmin>$sigs set 0 java.lang.String
    wsadmin>$sigs set 1 java.lang.Boolean
```

- Invoke the method. Use **invoke_jmx**, because it has a parameter.

```
wsadmin>$AdminControl invoke_jmx $perfOName
setInstrumentationLevel $params $sigs
```

This command does not return anything.

Note: The PMI level string can be as simple as *pmi=level* (where level is n, l, m, h, or x), or something like *module1=level1:module2=level2:module3=level3* with the same format shown in the string returned from `getInstrumentationLevelString`.

Invoke getStatsString(ObjectName, Boolean) operation

If you know the MBean ObjectName, you can invoke the method by passing the right parameters. As an example, JVM MBean is used here.

- Get MBean query string. For example, JVM MBean.

```
 wsadmin>set jvmName [$AdminControl completeObjectName
type=JVM,*]
```

- Set parameters.

```
wsadmin>set params [java::new {java.lang.Object[]} 2]

wsadmin>$params set 0 [$AdminControl makeObjectName $jvmName]
wsadmin>$params set 1 [java::new java.lang.Boolean true]
```

- Set signatures.

```
wsadmin>set sigs  [java::new {java.lang.String[]} 2]
    wsadmin>$sigs set 0 javax.management.ObjectName wsadmin>$sigs
    set 1 java.lang.Boolean
```

- Invoke method.

```
 wsadmin>$AdminControl invoke_jmx $perfOName getStatsString
$params $sigs
```

This command returns the following:

```
{Description jvmRuntimeModule.desc} {Descriptor {{Node wenjianpc}
{Server server
1} {Module jvmRuntimeModule} {Name jvmRuntimeModule} {Type
MODULE}}} {Level 7} {
Data {{{Id 4} {Descriptor {{Node wenjianpc} {Server server1}
{Module jvmRuntimeM
odule} {Name jvmRuntimeModule} {Type DATA}}} {PmiDataInfo {{Name
jvmRuntimeModul
e.upTime} {Id 4} {Description jvmRuntimeModule.upTime.desc} {Level
1} {Comment {
The amount of time in seconds the JVM has been running}}
{SubmoduleName null} {T
```

```
ype 2} {Unit unit.second} {Resettable false}}} {Time 1033670422282}
{Value {Coun
t 638} }} {{Id 3} {Descriptor {{Node wenjianpc} {Server server1}
{Module jvmRunt
imeModule} {Name jvmRuntimeModule} {Type DATA}}} {PmiDataInfo
{{Name jvmRuntimeM
odule.usedMemory} {Id 3} {Description
jvmRuntimeModule.usedMemory.desc} {Level 1
} {Comment {Used memory in JVM runtime}} {SubmoduleName null} {Type
2} {Unit uni
t.kbyte} {Resettable false}}} {Time 1033670422282} {Value {Count
66239} }} {{Id
2} {Descriptor {{Node wenjianpc} {Server server1} {Module
jvmRuntimeModule} {Nam
e jvmRuntimeModule} {Type DATA}}} {PmiDataInfo {{Name
jvmRuntimeModule.freeMemor
y} {Id 2} {Description jvmRuntimeModule.freeMemory.desc} {Level 1}
{Comment {Fre
e memory in JVM runtime}} {SubmoduleName null} {Type 2} {Unit
unit.kbyte} {Reset
table false}}} {Time 1033670422282} {Value {Count 34356} }} {{Id 1}
{Descriptor
{{Node wenjianpc} {Server server1} {Module jvmRuntimeModule} {Name
jvmRuntimeMod
ule} {Type DATA}}} {PmiDataInfo {{Name
jvmRuntimeModule.totalMemory} {Id 1} {Des
cription jvmRuntimeModule.totalMemory.desc} {Level 7} {Comment
{Total memory in
JVM runtime}} {SubmoduleName null} {Type 5} {Unit unit.kbyte}
{Resettable false}
}} {Time 1033670422282} {Value {Current 100596} {LowWaterMark
38140} {HighWaterM
ark 100596} {MBean 38140.0} }}}}
```

Invoke getStatsString (ObjectName, String, Boolean) operation

This operation takes an additional String parameter, and it is used for PMI modules that do not have matching MBeans. In this case, the parent MBean is used with a String name representing the PMI module. The String names available in an MBean can be found by invoking listStatMemberNames. For example, *beanModule* is a logic module aggregating PMI data over all Enterprise JavaBeans, but there is no MBean for beanModule. Therefore, you can pass server MBean ObjectName and a String (*beanModule*) to get PMI data in beanModule.

- Get MBean query string. For example, server MBean

  ```
  wsadmin>set mySrvName [$AdminControl completeObjectName
      type=Server,name=server1,
      node=wenjianpc,*]
  ```

- Set parameters.

  ```
  wsadmin>set params [java::new {java.lang.Object[]} 3]

  wsadmin>$params set 0 [$AdminControl makeObjectName $mySrvName]

  wsadmin>$params set 1 [java::new java.lang.String beanModule]
      wsadmin>$params set 2 [java::new java.lang.Boolean true]
  ```

- Set signatures.

  ```
  wsadmin>set sigs  [java::new {java.lang.String[]} 3]

  wsadmin>$sigs set 0 javax.management.ObjectName

  wsadmin>$sigs set 1 java.lang.String
      wsadmin>$sigs set 2 java.lang.Boolean
  ```

- Invoke method.

  ```
  wsadmin>$AdminControl invoke_jmx $perfOName getStatsString
  $params $sigs
  ```

This command returns PMI data in all the Enterprise JavaBeans within the BeanModule hierarchy because the recursive flag is set to true.

Note: This method is used to get stats data for the PMI modules that do not have direct MBean mappings.

Invoke listStatMemberNames operation

- Get MBean queryString. For example, Server.

```
wsadmin>set mySrvName [$AdminControl completeObjectName
    type=Server,name=server1,
node=wenjianpc,*]
```

- Set parameter.

```
wsadmin>set params [java::new {java.lang.Object[]} 1]
wsadmin>$params set 0 [$AdminControl makeObjectName
$mySrvName]
```

- Set signatures.

```
wsadmin>set sigs  [java::new {java.lang.String[]} 1]
wsadmin>$sigs set 0 javax.management.ObjectName
    wsadmin>$AdminControlinvoke_jmx $perfOName
    listStatMemberNames $params $sigs
```

This command returns the PMI module and submodule names, which have no direct MBean mapping. The names are seperated by a space ″ ″. You can then use the name as the String parameter in getStatsString method. For example:

```
beanModule connectionPoolModule j2cModule servletSessionsModule
threadPoolModule
webAppModule
```

Customize and run the following example Jython script with the wsadmin tool to obtain PMI data:

```
print "\n------------------------------------------------------------------------------------------------------- "
print "Obtain the Perf MBean ObjectName"
print "------------------------------------------------------------------------------------------------------- "
perfName = AdminControl.completeObjectName ('type=Perf,*')
perfOName = AdminControl.makeObjectName (perfName)
print perfOName
print "------------------------------------------------------------------------------------------------------- \n"


print "\n------------------------------------------------------------------------------------------------------- "
print "Invoke getStatisticSet operation "
print "------------------------------------------------------------------------------------------------------- "
print AdminControl.invoke (perfName, 'getStatisticSet')
print "------------------------------------------------------------------------------------------------------- \n"


print "\n------------------------------------------------------------------------------------------------------- "
print "Invoke setStatisticSet operation"
print "------------------------------------------------------------------------------------------------------- "
params = ['extended']

sigs  = ['java.lang.String']

print AdminControl.invoke_jmx (perfOName, 'setStatisticSet', params, sigs)
print "------------------------------------------------------------------------------------------------------- \n"


print "\n------------------------------------------------------------------------------------------------------- "
print "Invoke getConfig operation"
print "------------------------------------------------------------------------------------------------------- "
jvmName = AdminControl.completeObjectName ('type=JVM,*')

params = [AdminControl.makeObjectName (jvmName)]
```

```
sigs = ['javax.management.ObjectName']

print AdminControl.invoke_jmx (perfONName, 'getConfig', params, sigs)
print "------------------------------------------------------------------------------------------------------------ \n"


print "\n------------------------------------------------------------------------------------------------------------ "
print "Invoke getCustomSetString operation"
print "------------------------------------------------------------------------------------------------------------ "
print AdminControl.invoke (perfName, 'getCustomSetString')
print "------------------------------------------------------------------------------------------------------------ \n"


print "\n------------------------------------------------------------------------------------------------------------ "
print "Invoke setCustomSetString operation"
print "------------------------------------------------------------------------------------------------------------ "
params = ['jvmRuntimeModule=1,2,3,4', java.lang.Boolean ('false')]

sigs  = ['java.lang.String', 'java.lang.Boolean']

print AdminControl.invoke_jmx (perfONName, 'setCustomSetString', params, sigs)
print "------------------------------------------------------------------------------------------------------------ \n"


print "\n------------------------------------------------------------------------------------------------------------ "
print "Invoke getStatsObject operation"
print "------------------------------------------------------------------------------------------------------------ "
jvmName = AdminControl.completeObjectName ('type=JVM,*')

params = [AdminControl.makeObjectName (jvmName), java.lang.Boolean ('false')]

sigs = ['javax.management.ObjectName', 'java.lang.Boolean']

print AdminControl.invoke_jmx (perfONName, 'getStatsObject', params, sigs)
print "------------------------------------------------------------------------------------------------------------ \n"


print "\n------------------------------------------------------------------------------------------------------------ "
print "Invoke getInstrumentationLevelString operation"
print "------------------------------------------------------------------------------------------------------------ "
print AdminControl.invoke (perfName, 'getInstrumentationLevelString')
print "------------------------------------------------------------------------------------------------------------ \n"


print "\n------------------------------------------------------------------------------------------------------------ "
print "Invoke setInstrumentationLevel operation - enable/disable PMI counters "
print "------------------------------------------------------------------------------------------------------------ "
params = ['pmi=1', java.lang.Boolean ('true')]

sigs = ['java.lang.String', 'java.lang.Boolean']

print AdminControl.invoke_jmx (perfONName, 'setInstrumentationLevel', params, sigs)
print "------------------------------------------------------------------------------------------------------------ \n"


print "\n------------------------------------------------------------------------------------------------------------ "
print "Invoke getStatsString(ObjectName, Boolean) operation"
print "------------------------------------------------------------------------------------------------------------ "
jvmName = AdminControl.completeObjectName ('type=JVM,*')

params = [AdminControl.makeObjectName (jvmName), java.lang.Boolean ('true')]

sigs = ['javax.management.ObjectName', 'java.lang.Boolean']

print AdminControl.invoke_jmx (perfONName, 'getStatsString', params, sigs)
print "------------------------------------------------------------------------------------------------------------ \n"
```

```
print "\n------------------------------------------------------------------------------------------------------------ "
print "Invoke getStatsString (ObjectName, String, Boolean) operation"
print "------------------------------------------------------------------------------------------------------------ "
mySrvName = AdminControl.completeObjectName ('type=Server,name=server1,node=wcsNode,*')

params = [AdminControl.makeObjectName (mySrvName),
          'beanModule',
          java.lang.Boolean ('true')]

sigs = ['javax.management.ObjectName',
        'java.lang.String',
        'java.lang.Boolean')

print AdminControl.invoke_jmx (perfOName, 'getStatsString', params, sigs)
print "------------------------------------------------------------------------------------------------------------ \n"


print "\n------------------------------------------------------------------------------------------------------------ "
print "Invoke listStatMemberNames operation"
print "------------------------------------------------------------------------------------------------------------ "
mySrvName = AdminControl.completeObjectName ('type=Server,name=server1,node=wcsNode,*')

params = [AdminControl.makeObjectName (mySrvName)]

sigs = ['javax.management.ObjectName']

print AdminControl.invoke_jmx (perfOName, 'listStatMemberNames', params, sigs)
print "------------------------------------------------------------------------------------------------------------ \n"
```

Obtaining a list of performance counters from the command line

Obtain a list of performance counters from the command line.

Before you begin

The application server must be running when you obtain a list of performance counters from the command line.

About this task

You can obtain a list of performance counters from the command line.

1. Bring up a command line window.
2. Type the following commands:

```
set perf [$AdminControl completeObjectName type=Perf,*]
set perfON [$AdminControl makeObjectName $perf]
set params [java::new {java.lang.Object[]} 1]
$params set 0 [java::new java.util.Locale "en-US"]
set sigs  [java::new {java.lang.String[]} 1]
$sigs set 0 java.util.Locale
set out [java::cast {com.ibm.websphere.pmi.PmiModuleConfig[]}
[$AdminControl invoke_jmx $perfON getConfigs $params $sigs]]
for {set i 0} {$i < [$out length]} {incr i 1} { puts [[$out get $i] toString] }
```

3. Results similar to the following example are returned:

```
Stats type=systemModule, Description=The system performance data from
the operating system.

{name=FreeMemory, ID=3, type=CountStatistic, description=A
snapshot of
free memory (in KB)., unit=N/A, level=low, statisticSet=all,
resettable=false, aggregatable=true, zosAggregatable=true}
```

```
{name=CPUUsageSinceServerStarted, ID=2, type=AverageStatistic,
description=The average CPU utilization since the server was
started.,
unit=N/A, level=medium, statisticSet=extended, resettable=true,
aggregatable=true, zosAggregatable=true}

{name=CPUUsageSinceLastMeasurement, ID=1, type=CountStatistic,
description=The average CPU utilization since the last query.,
unit=N/A,
level=low, statisticSet=basic, resettable=false,
aggregatable=true,
zosAggregatable=true
```

Enabling the Java virtual machine profiler data

Use the Java virtual machine profiler to enable the collection of information.

About this task

Use the Java Virtual Machine Tool Interface (JVMTI) to collect data about garbage collection and thread states from the JVM that runs the application server. When JVMTI is enabled, a collection of JVMTI-specific statistics can be enabled through the predefined All statistic set or through the use of a Custom statistic set.

To enable JVMTI data reporting for each individual application server:

1. Open the administrative console.
2. Click **Servers > Server Types > WebSphere application servers** in the administrative console navigation tree.
3. Click the application server for the JVM profiler that needs enabling.
4. In the Server Infrastructure section, click **Java and process management > Process definition** , click **Control**, and then click **Java virtual machine**.
5. Type *-agentlib:pmiJvmtiProfiler* in the **Generic JVM arguments** field. In a 64-bit environment, type *-agentlib:pmiJvmtiProfiler64*.
6. Click **Apply** or **OK**.
7. Click **Save**.
8. Click **Servers > Server Types > WebSphere application servers** in the administrative console navigation tree.
9. Click the application server for the JVM profiler that needs enabling.
10. Click the **Configuration** tab. When in the **Configuration** tab, settings apply when the server is restarted. When in the **Runtime** Tab, settings apply immediately. Performance Monitoring Infrastructure (PMI) can be enabled only in the **Configuration** tab.
11. Click **Performance Monitoring Infrastructure** under Performance.
12. Select the **Enable Performance Monitoring Infrastructure (PMI)** check box.
13. Click **Custom** and select **JVM Runtime** on the left-side tree.
14. Click a JVM profiler group under **JVM Runtime** and enable or disable the statistic on the right-side table. Go back to the main PMI configuration page, by clicking PMI.
15. Click **Apply** or **OK**.
16. Click **Save**.
17. Start the application server, or restart the application server if it is currently running.
18. Refresh the Tivoli Performance Viewer if you are using it. The changes you make will not take effect until you restart the application server.

Java virtual machine profiling

Use Java virtual machine profiling to gather data about your system for performance analysis.

The Java virtual machine Tool Interface (JVMTI) is a native programming interface that provides tools the ability to inspect the state of the Java virtual machine (JVM).

JVMTI provides the ability to collect information about the JVM that runs the application server. The Tivoli Performance Viewer leverages these interfaces to enable more comprehensive performance analysis.

JVMTI is a two-way function call interface between the JVM and an in-process profiler agent. The JVM notifies the profiler agent of various events, for example, garbage collection and thread starts. The profiler agent activates or deactivates specific event notifications that are based on the needs of the profiler.

JVMTI supports partial profiling, by enabling you to choose which types of profiling information to collect and to select certain subsets of the time during which the JVM API is active. JVMTI moderately increases the performance impact. Therefore, it is recommended that you use JVMTI monitoring to help diagnose application problems only.

Developing your own monitoring applications

You can use the Performance Monitoring Infrastructure (PMI) interfaces to develop your own applications to collect and display performance information.

About this task

There are three such interfaces - a Java Machine Extension (JMX)-based interface, a PMI client interface, and a servlet interface. All three interfaces return the same underlying data.

The JMX interface is accessible through the AdminClient tool as described in “Using the JMX interface to develop your own monitoring application” on page 204. The PMI client interface is a Java interface. The servlet interface is perhaps the simplest, requiring minimal programming, as the output is XML.

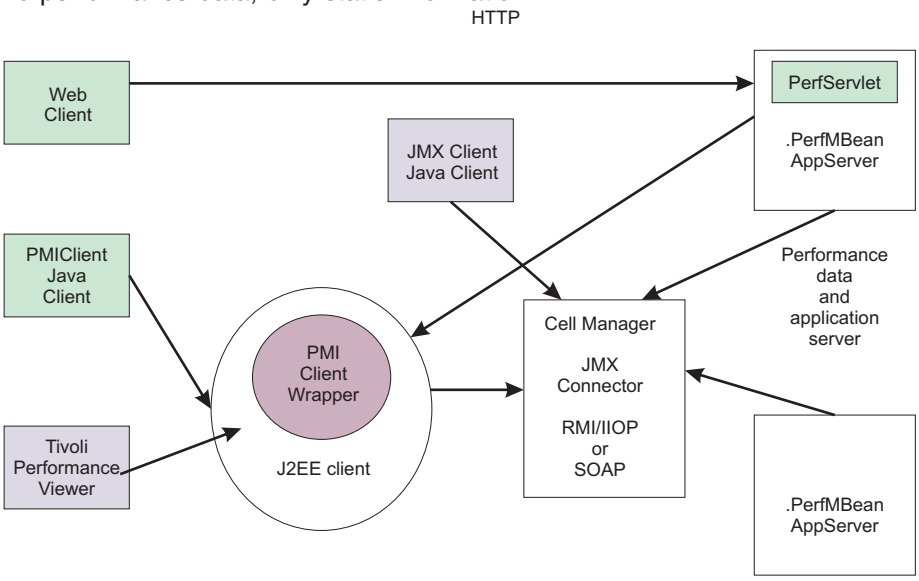
1. “Using PMI client to develop your monitoring application (deprecated)” on page 189.
2. “Retrieving performance data with PerfServlet” on page 200
3. “Compiling your monitoring applications” on page 221
4. “Running your new monitoring applications” on page 221
5. “Using the JMX interface to develop your own monitoring application” on page 204.
6. “Developing PMI interfaces (Version 4.0) (deprecated)” on page 220.

PMI client interface (deprecated)

The data provided by the Performance Monitoring Infrastructure (PMI) client interface is documented here.

Access to the PMI client interface data is provided in a hierarchical structure. Descending from the object are node information objects, module information objects, CpdCollection objects and CpdData objects. Using Version 5.0, you will get Stats and Statistic objects. The node and server information objects contain

no performance data, only static information.



Each time a client retrieves performance data from a server, the data is returned in a subset of this structure; the form of the subset depends on the data retrieved. You can update the entire structure with new data, or update only part of the tree, as needed.

The JMX statistic data model is supported, as well as the existing CPD data model from Version 4.0. When you retrieve performance data using the Version 5.0 PMI client API, you get the Stats object, which includes Statistic objects and optional sub-Stats objects. When you use the Version 4.0 PMI client API to collect performance data, you get the CpdCollection object, which includes the CpdData objects and optional sub-CpdCollection objects.

The following are additional Performance Monitoring Infrastructure (PMI) interfaces:
- BoundaryStatistic
- BoundedRangeStatistic
- CountStatistic
- MBeanStatDescriptor
- MBeanLevelSpec
- New Methods in PmiClient
- RangeStatistic
- Stats
- Statistic
- TimeStatistic

The following PMI interfaces introduced in Version 4.0 are also supported:
- CpdCollection
- CpdData
- CpdEventListener and CpdEvent
- CpdFamily class
- CpdValue
  - CpdLong
  - CpdStat
  - CpdLoad
- PerfDescriptor
- PmiClient class

The CpdLong maps to CountStatistic; CpdStat maps to Time Statistic; CpdCollection maps to Stats; and CpdLoad maps to RangeStatistic and BoundedRangeStatistic.

Note: Version 4.0 PmiClient APIs are supported in this version, however, there are some changes. The data hierarchy is changed in some PMI modules, notably the enterprise bean module and HTTP sessions module. If you have an existing PmiClient application, and you want to run it against Version 5.0, you might have to update the PerfDescriptor(s) based on the new PMI data hierarchy. Also, the getDataName and getDataId methods in PmiClient are changed to be non-static methods in order to support multiple WebSphere Application Server versions. You might have to update your existing application which uses these two methods.

Using PMI client to develop your monitoring application (deprecated)

You can use the Performance Monitoring Infrastructure (PMI) interfaces to develop your own applications to collect and display performance information.

About this task

The following is the programming model for Performance Monitoring Infrastructure (PMI) client:

1. Create an instance of PmiClient. This is used for all subsequent method calls.
2. Optional: You can create your own MBeans. Refer to Extending the WebSphere Application Server administrative system with custom MBeans.
3. Call the listNodes() and listServers(nodeName) methods to find all the nodes and servers in the WebSphere Application Server domain.
4. Call listMBeans and listStatMembers to get all the available MBeans and MBeanStatDescriptors.
5. Call the getStats method to get the Stats object for the PMI data.
6. Optional: The client can also call setStatLevel or getStatLevel to set and get the monitoring level. Use the MBeanLevelSpec objects to set monitoring levels.

Performance Monitoring Infrastructure client (WebSphere Version 4.0)

A Performance Monitoring Infrastructure (PMI) client is an application that receives PMI data from servers and processes this data.

In WebSphere Application Server Version 4.0, PmiClient API takes PerfDescriptor(s) and returns PMI data as a CpdCollection object. Each CpdCollection could contain a list of CpdData, which has a CpdValue of the following types:

- CpdLong
- CpdStat
- CpdLoad

Version 4.0 PmiClient APIs are supported in this version, however, there are some changes. The data hierarchy is changed in some PMI modules, notably the enterprise bean module and HTTP sessions module. If you have an existing PmiClient application, and you want to run it against WebSphere Application Server Version 5.0 or above, you might have to update the PerfDescriptor(s) based on the new PMI data hierarchy. Also, the getDataName and getDataId methods in PmiClient are changed to be non-static methods in order to support multiple WebSphere Application Server versions. You might have to update your existing application which uses these two methods.

Example: Performance Monitoring Infrastructure client (Version 4.0)

The following is a list of example Performance Monitoring Infrastructure (PMI) client code:

```
/**
 * This is a sample code to show how to use PmiClient to collect PMI data.
 * You will need to use adminconsole to set instrumentation level (a level other
 * than NONE) first.
 *
 * <p>
 * End-to-end code path:
 *     PmiTester -> PmiClient -> AdminServer -> appServer
```

```
 */


package com.ibm.websphere.pmi;

import com.ibm.websphere.pmi.*;
import com.ibm.websphere.pmi.server.*;
import com.ibm.websphere.pmi.client.*;
import com.ibm.ws.pmi.server.*;
import com.ibm.ws.pmi.perfServer.*;
import com.ibm.ws.pmi.server.modules.*;
import com.ibm.ws.pmi.wire.*;
import java.util.ArrayList;


/**
 * Sample code to use PmiClient API and get CpdData/CpdCollection objects.
 *
 */
public class PmiTester implements PmiConstants {

    /** a test driver:
     * @param args[0] - node name
     * @param args[1] - port number, optional, default is 2809
     * @param args[2] - connector type, default is RMI
     * @param args[3] - verion (AE, AEs, WAS50), default is WAS50
     *
     */
    public static void main(String[] args) {
        String hostName = null;
        String portNumber = "2809";
        String connectorType = "RMI";
        String version = "WAS50";

        if (args.length < 1) {
            System.out.println("Usage: <host> [<port>] [<connectorType>]
[<version>]");
            return;
        }

        if(args.length >= 1)
            hostName = args[0];
        if(args.length >= 2)
            portNumber = args[1];
        if (args.length >=3)
            connectorType = args[2];
        if (args.length >=4)
            version = args[3];

        try {
            PmiClient pmiClnt = new PmiClient(hostName, portNumber,
version, false, connectorType);

            // uncomment it if you want debug info
            //pmiClnt.setDebug(true);

            // get all the node PerfDescriptor in the domain
            PerfDescriptor[] nodePds = pmiClnt.listNodes();

            if(nodePds == null) {
                System.out.println("no nodes");
                return;
            }

            // get the first node
            String nodeName = nodePds[0].getName();
            System.out.println("after listNodes: " + nodeName);
```

```
            //list all the servers on the node
            PerfDescriptor[] serverPds = pmiClnt.listServers(nodePds[0].getName());
            if(serverPds == null || serverPds.length == 0) {
                System.out.println("NO app server in node");
                return;
            }

            // print out all the servers on that node
            for(int j=0; j<serverPds.length; j++) {
                System.out.println("server " + j + ": " + serverPds[j].getName());
            }

            for(int j=0; j<serverPds.length; j++) {
                System.out.println("server " + j + ": " + serverPds[j].getName());

                // Option: you can call createPerfLevelSpec and then
setInstrumentationLevel to set the level
                // for each server if you want. For example, to set all
the modules to be LEVEL_HIGH for the server j,
                // uncomment the following.
                // PerfLevelSpec[] plds = new PerfLevelSpec[1];
                // plds[0] = pmiClnt.createPerfLevelSpec(null, LEVEL_HIGH);
                // pmiClnt.setInstrumentationLevel(serverPds[j].getNodeName(),
serverPds[j].getServerName(), plds, true);

                // First, list the PerfDescriptor in the server
                PerfDescriptor[] myPds = pmiClnt.listMembers(serverPds[j]);

                // check returned PerfDescriptor
                if(myPds == null) {
                    System.out.println("null from listMembers");
                    continue;
                }

                // you can  add the pds in which you are interested to PerfDescriptorList
                PerfDescriptorList pdList = new PerfDescriptorList();
                for(int i=0; i<myPds.length; i++) {
                    // Option 1: you can recursively call listMembers for each myPds
                    //               and find the one you are interested. You can call
listMembers
                    //               until individual data level and after that level
you will null from listMembers.
                    // e.g., PerfDescriptor[] nextPds = pmiClnt.listMembers(myPds[i]);

                    // Option 2: you can filter these pds  before adding to pdList
                    System.out.println("add to pdList: " + myPds[i].getModuleName());
                    pdList.addDescriptor(myPds[i]);
                    if( i % 2 == 0)
                        pmiClnt.add(myPds[i]);
                }

                // call gets method to get the CpdCollection[] corresponding to pdList
                CpdCollection[] cpdCols = pmiClnt.gets(pdList, true);

                if(cpdCols == null) {
                    // check error
                    if(pmiClnt.getErrorCode() >0)
                        System.out.println(pmiClnt.getErrorMessage());
                    continue;
                }

                for(int i=0; i<cpdCols.length; i++) {
                    // simple print them
                    //System.out.println(cpdCols[i].toString());

                    // Or call processCpdCollection to get each data
```

```
                    processCpdCollection(cpdCols[i], "");
                }

                // Or call gets() method to add the CpdCollection[] for whatever
there by calling pmiClnt.add().
                System.out.println("\n\n\n ---- get data using gets(true) ----- ");
                cpdCols = pmiClnt.gets(true);

                if(cpdCols == null) {
                    // check error
                    if(pmiClnt.getErrorCode() >0)
                        System.out.println(pmiClnt.getErrorMessage());
                    continue;
                }

                for(int i=0; i<cpdCols.length; i++) {
                    // simple print out the whole collection
                    System.out.println(cpdCols[i].toString());

                    // Option: refer processCpdCollection to get each data
                }
            }

        }
        catch(Exception ex) {
            System.out.println("Exception calling CollectorAE");
            ex.printStackTrace();
        }
    }

    /**
     * show the methods to retrieve individual data
     */
    private static void processCpdCollection(CpdCollection cpdCol, String indent) {
        CpdData[] dataList = cpdCol.dataMembers();
        String myindent = indent;

        System.out.println("\n" + myindent + "--- CpdCollection "
+ cpdCol.getDescriptor().getName() + " ---");
        myindent += "   ";
        for(int i=0; i<dataList.length; i++) {
            if (dataList[i] == null)
                continue;

            // if you want to get static info like name, description, etc
            PmiDataInfo dataInfo = dataList[i].getPmiDataInfo();
            // call getName(), getDescription() on dataInfo;

            CpdValue cpdVal = dataList[i].getValue();
            if(cpdVal.getType() == TYPE_STAT) {
                CpdStat cpdStat = (CpdStat)cpdVal;
                double mean = cpdStat.mean();
                double sumSquares = cpdStat.sumSquares();
                int count = cpdStat.count();
                double total = cpdStat.total();
                System.out.println(myindent + "CpdData id=" + dataList[i].getId()
                                   + " type=stat mean=" + mean);
                // you can print more values like sumSquares, count,etc here
            }
            else if(cpdVal.getType() == TYPE_LOAD) {
                CpdLoad cpdLoad = (CpdLoad)cpdVal;
                long time = cpdLoad.getTime();
                double mean = cpdLoad.mean();
                double currentLevel = cpdLoad.getCurrentLevel();
                double integral = cpdLoad.getIntegral();
                double timeWeight = cpdLoad.getWeight();
                System.out.println(myindent + "CpdData id=" + dataList[i].getId()
```

```
                                + " type=load mean=" + mean + " currentLevel="
+ currentLevel);
                // you can print more values like sumSquares, count,etc here
            }
            else if(cpdVal.getType() == TYPE_LONG) {
                CpdValue cpdLong = (CpdValue)cpdVal;
                long value = (long)cpdLong.getValue();
                System.out.println(myindent + "CpdData id=" + dataList[i].getId()
                                + " type=long value=" + value);
            }
            else if(cpdVal.getType() == TYPE_DOUBLE) {
                CpdValue cpdDouble = (CpdValue)cpdVal;
                double value = cpdDouble.getValue();
                System.out.println(myindent + "CpdData id=" + dataList[i].getId()
                                + " type=double value=" + value);
            }
            else if(cpdVal.getType() == TYPE_INT) {
                CpdValue cpdInt = (CpdValue)cpdVal;
                int value = (int)cpdInt.getValue();
                System.out.println(myindent + "CpdData id=" + dataList[i].getId()
                                + " type=int value=" + value);
            }
        }

        // recursively go through the subcollection
        CpdCollection[] subCols = cpdCol.subcollections();
        for(int i=0; i<subCols.length; i++) {
            processCpdCollection(subCols[i], myindent);
        }
    }

    /**
     * show the methods to navigate CpdCollection
     */
    private static void report(CpdCollection col) {
        System.out.println("\n\n");
        if(col==null) {
            System.out.println("report: null CpdCollection");
            return;
        }
        System.out.println("report - CpdCollection ");
        printPD(col.getDescriptor());
        CpdData[] dataMembers = col.dataMembers();
        if(dataMembers != null) {
            System.out.println("report CpdCollection: dataMembers is "
+ dataMembers.length);
            for(int i=0; i<dataMembers.length; i++) {
                CpdData data = dataMembers[i];
                printPD(data.getDescriptor());
            }
        }
        CpdCollection[] subCollections = col.subcollections();
        if(subCollections != null) {
            for(int i=0; i<subCollections.length; i++) {
                report(subCollections[i]);
            }
        }
    }

    private static void printPD(PerfDescriptor pd) {
        System.out.println(pd.getFullName());
    }

}
```

Example: Performance Monitoring Infrastructure client with new data structure

This page provides example code using Performance Monitoring Infrastructure (PMI) client with the new data structure.

The following is example code using Performance Monitoring Infrastructure (PMI) client data structure:

```
import com.ibm.websphere.pmi.*;
import com.ibm.websphere.pmi.stat.*;
import com.ibm.websphere.pmi.client.*;
import com.ibm.websphere.management.*;
import com.ibm.websphere.management.exception.*;
import java.util.*;
import javax.management.*;
import java.io.*;

/**
 * Sample code to use PmiClient API (new JMX-based API) and
get Statistic/Stats objects.
 */

public class PmiClientTest implements PmiConstants {

    static PmiClient pmiClnt = null;
    static String nodeName = null;
    static String serverName = null;
    static String portNumber = null;
    static String connectorType = null;
    static boolean success = true;


    /**
     * @param args[0] host
     * @param args[1] portNumber, optional, default is 2809
     * @param args[2] connectorType, optional, default is RMI connector
     * @param args[3]serverName, optional, default is the first server found
     */
    public static void main(String[] args) {

        try {

            if(args.length > 1) {
                System.out.println("Parameters: host [portNumber]
[connectorType] [serverName]");
                return;
            }

            // parse arguments and create an instance of PmiClient
            nodeName = args[0];

            if (args.length > 1)
            portNumber = args[1];

            if (args.length > 2)
            connectorType = args[2];

            // create an PmiClient object
            pmiClnt = new PmiClient(nodeName, portNumber, "WAS50", false, connectorType);

            // Uncomment it if you want to debug any problem
            //pmiClnt.setDebug(true);

            // update nodeName to be the real host name
            // get all the node PerfDescriptor in the domain
               PerfDescriptor[] nodePds = pmiClnt.listNodes();
               if(nodePds == null) {
               System.out.println("no nodes");
               return;
```

```
                        }
// get the first node
   nodeName = nodePds[0].getName();
 System.out.println("use node " + nodeName);

 if (args.length == 4)
     serverName = args[3];
 else { // find the server you want to get PMI data
     // get all servers on this node
     PerfDescriptor[] allservers = pmiClnt.listServers(nodeName);
     if (allservers == null || allservers.length == 0) {
         System.out.println("No server is found on node " + nodeName);
         System.exit(1);
     }

     // get the first server on the list. You may want to get a different server
     serverName = allservers[0].getName();
     System.out.println("Choose server " + serverName);
 }

 // get all MBeans
 ObjectName[] onames = pmiClnt.listMBeans(nodeName, serverName);

 // Cache the MBeans we are interested
 ObjectName perfOName = null;
 ObjectName serverOName = null;
 ObjectName wlmOName = null;
 ObjectName ejbOName = null;
 ObjectName jvmOName = null;
 ArrayList myObjectNames = new ArrayList(10);

 // get the MBeans we are interested in
 if(onames != null) {
     System.out.println("Number of MBeans retrieved= " + onames.length);
     AttributeList al;
     ObjectName on;
     for(int i=0; i<onames.length; i++) {
         on = onames[i];
         String type = on.getKeyProperty("type");

         // make sure PerfMBean is there.
         // Then randomly pick up some MBeans for the test purpose
         if(type != null && type.equals("Server"))
             serverOName = on;
         else if(type != null && type.equals("Perf"))
             perfOName = on;
         else if(type != null && type.equals("WLM")) {
             wlmOName = on;
         }
         else if(type != null && type.equals("EntityBean")) {
             ejbOName = on;

             // add all the EntityBeans to myObjectNames
             myObjectNames.add(ejbOName);  // add to the list
         }
         else if(type != null && type.equals("JVM")) {
             jvmOName = on;
         }
     }

     // set monitoring level for SERVER MBean
     testSetLevel(serverOName);

     // get Stats objects
     testGetStats(myObjectNames);

     // if you know the ObjectName(s)
```

```
            testGetStats2(new ObjectName[]{jvmONname, ejbONname});

            // assume you are only interested in a server data in WLM MBean,
            // then you will need to use StatDescriptor and MBeanStatDescriptor
            // Note that wlmModule is only available in ND version
            StatDescriptor sd = new StatDescriptor(new String[] {"wlmModule.server"});
            MBeanStatDescriptor msd = new MBeanStatDescriptor(wlmONname, sd);
            Stats wlmStat = pmiClnt.getStats(nodeName, serverName, msd, false);
            if (wlmStat != null)
                System.out.println("\n\n WLM server data\n\n + " + wlmStat.toString());
            else
                System.out.println("\n\n No WLM server data is availalbe.");

            // how to find all the MBeanStatDescriptors
            testListStatMembers(serverONname);

            // how to use update method
            testUpdate(jvmONname, false, true);
        }
        else {
            System.out.println("No ObjectNames returned from Query" );
        }

    }
    catch(Exception e) {
        new AdminException(e).printStackTrace();
        System.out.println("Exception = " +e);
        e.printStackTrace();
        success = false;
    }


    if(success)
        System.out.println("\n\n All tests are passed");
    else
        System.out.println("\n\n Some tests are failed. Check for the exceptions");

}

/**
 * construct an array from the ArrayList
 */
private static MBeanStatDescriptor[] getMBeanStatDescriptor(ArrayList msds) {
    if(msds == null || msds.size() == 0)
        return null;

    MBeanStatDescriptor[] ret = new MBeanStatDescriptor[msds.size()];
    for(int i=0; i<ret.length; i++)
        if(msds.get(i) instanceof ObjectName)
            ret[i] = new MBeanStatDescriptor((ObjectName)msds.get(i));
        else
            ret[i] = (MBeanStatDescriptor)msds.get(i);
    return ret;
}

/**
 * Sample code to navigate and display the data value from the Stats object.
 */
private static void processStats(Stats stat) {
    processStats(stat, "");
}

/**
 * Sample code to navigate and display the data value from the Stats object.
 */
private static void processStats(Stats stat, String indent) {
    if(stat == null)  return;
```

```
        System.out.println("\n\n");

        // get name of the Stats
        String name = stat.getName();
        System.out.println(indent + "stats name=" + name);

        // Uncomment the following lines to list all the data names
        /*
        String[] dataNames = stat.getStatisticNames();
        for (int i=0; i<dataNames.length; i++)
            System.out.println(indent + "    " + "data name=" + dataNames[i]);
        System.out.println("\n");
        */

        // list all datas
        com.ibm.websphere.management.statistics.Statistic[] allData = stat.getStatistics();

        // cast it to be PMI's Statistic type so that we can have get more
        Statistic[] dataMembers = (Statistic[])allData;
        if(dataMembers != null) {
            for(int i=0; i<dataMembers.length; i++)  {
                System.out.print(indent + "    " + "data name="
+ PmiClient.getNLSValue(dataMembers[i].getName())
                                + ", description="
+ PmiClient.getNLSValue(dataMembers[i].getDescription())
                                + ", unit=" + PmiClient.getNLSValue(dataMembers[i].getUnit())
                                + ", startTime=" + dataMembers[i].getStartTime()
                                + ", lastSampleTime=" + dataMembers[i].getLastSampleTime());
                if(dataMembers[i].getDataInfo().getType() == TYPE_LONG) {
                    System.out.println(", count="
+ ((CountStatisticImpl)dataMembers[i]).getCount());
                }
                else if(dataMembers[i].getDataInfo().getType() == TYPE_STAT) {
                    TimeStatisticImpl data = (TimeStatisticImpl)dataMembers[i];
                    System.out.println(", count=" + data.getCount()
                                       + ", total=" + data.getTotal()
                                       + ", mean=" + data.getMean()
                                       + ", min=" + data.getMin()
                                       + ", max=" + data.getMax());
                }
                else if(dataMembers[i].getDataInfo().getType() == TYPE_LOAD) {
                    RangeStatisticImpl data = (RangeStatisticImpl)dataMembers[i];
                    System.out.println(", current=" + data.getCurrent()
                                       + ", lowWaterMark=" + data.getLowWaterMark()
                                       + ", highWaterMark=" + data.getHighWaterMark()
                                       + ", integral=" + data.getIntegral()
                                       + ", avg=" + data.getMean());
                }
            }
        }

        // recursively for sub-stats
        Stats[] substats = (Stats[])stat.getSubStats();
        if(substats == null || substats.length == 0)
            return;
        for(int i=0; i<substats.length; i++) {
            processStats(substats[i], indent + "    ");
        }
    }

    /**
     * test set level and verify using get level
     */
    private static void testSetLevel(ObjectName mbean) {
        System.out.println("\n\n testSetLevel\n\n");
        try {
```

```
            // set instrumentation level to be high for the mbean
            MBeanLevelSpec spec = new MBeanLevelSpec(mbean, null, PmiConstants.LEVEL_HIGH);
            pmiClnt.setStatLevel(nodeName, serverName, spec, true);
            System.out.println("after setInstrumentaionLevel high on server MBean\n\n");

            // get all instrumentation levels
            MBeanLevelSpec[] mlss = pmiClnt.getStatLevel(nodeName, serverName, mbean, true);

            if(mlss == null)
                System.out.println("error: null from getInstrumentationLevel");
            else {
                for(int i=0; i<mlss.length; i++)
                    if(mlss[i] != null) {
                        // get the ObjectName, StatDescriptor,
and level out of MBeanStatDescriptor
                        int mylevel = mlss[i].getLevel();
                        ObjectName myMBean = mlss[i].getObjectName();
                        StatDescriptor mysd = mlss[i].getStatDescriptor();  // may be null
                        // Uncomment it to print all the mlss
                        //System.out.println("mlss " + i + ":, " + mlss[i].toString());
                    }
            }
        }
        catch(Exception ex) {
            new AdminException(ex).printStackTrace();
            ex.printStackTrace();
            System.out.println("Exception in testLevel");
            success = false;
        }
    }

    /**
     * Use listStatMembers method
     */
    private static void testListStatMembers(ObjectName mbean) {

        System.out.println("\n\ntestListStatMembers \n");
        // listStatMembers and getStats
        // From server MBean until the bottom layer.
        try {
            MBeanStatDescriptor[] msds = pmiClnt.listStatMembers(nodeName, serverName, mbean);
            if(msds == null) return;
            System.out.println(" listStatMembers for server MBean, num members
(i.e. top level modules) is " + msds.length);


            for(int i=0; i<msds.length; i++) {
                if(msds[i] == null)  continue;

                // get the fields out of MBeanStatDescriptor if you need them
                ObjectName myMBean = msds[i].getObjectName();
                StatDescriptor mysd = msds[i].getStatDescriptor();      // may be null

                // uncomment if you want to print them out
                //System.out.println(msds[i].toString());
            }

            for(int i=0; i<msds.length; i++) {
                if(msds[i] == null)  continue;
                System.out.println("\n\nlistStatMembers for msd=" + msds[i].toString());
                MBeanStatDescriptor[] msds2 =
pmiClnt.listStatMembers(nodeName, serverName, msds[i]);

                // you get msds2 at the second layer now and the
listStatMembers can be called recursively
                // until it returns now.
            }
```

```
        }
        catch(Exception ex) {
            new AdminException(ex).printStackTrace();
            ex.printStackTrace();
            System.out.println("Exception in testListStatMembers");
            success = false;
        }

    }

    /**
     * Test getStats method
     */
    private static void testGetStats(ArrayList mbeans) {
        System.out.println("\n\n testgetStats\n\n");
        try {
            Stats[] mystats = pmiClnt.getStats(nodeName,
serverName, getMBeanStatDescriptor(mbeans), true);

            // navigate each of the Stats object and get/display the value
            for(int k=0; k<mystats.length; k++) {
                processStats(mystats[k]);
            }

        }
        catch(Exception ex) {
            new AdminException(ex).printStackTrace();
            ex.printStackTrace();
            System.out.println("exception from testGetStats");
            success = false;
        }
    }

    /**
     * Test getStats method
     */
    private static void testGetStats2(ObjectName[] mbeans) {
        System.out.println("\n\n testGetStats2\n\n");
        try {
            Stats[] statsArray = pmiClnt.getStats(nodeName, serverName, mbeans, true);

            // You can call toString to simply display all the data
            if(statsArray != null) {
                for(int k=0; k<statsArray.length; k++)
                    System.out.println(statsArray[k].toString());
            }
            else
                System.out.println("null stat");
        }
        catch(Exception ex) {
            new AdminException(ex).printStackTrace();
            ex.printStackTrace();
            System.out.println("exception from testGetStats2");
            success = false;
        }
    }

    /**
     * test update method
     */
    private static void testUpdate(ObjectName oName, boolean keepOld,
boolean recursiveUpdate) {
        System.out.println("\n\n testUpdate\n\n");
        try {
            // set level to be NONE
            MBeanLevelSpec spec = new MBeanLevelSpec(oName, null, PmiConstants.LEVEL_NONE);
```

```
            pmiClnt.setStatLevel(nodeName, serverName, spec, true);


            // get data now - one is non-recursive and the other is recursive
            Stats stats1 = pmiClnt.getStats(nodeName, serverName, oName, false);
            Stats stats2 = pmiClnt.getStats(nodeName, serverName, oName, true);

            // set level to be HIGH
            spec = new MBeanLevelSpec(oName, null, PmiConstants.LEVEL_HIGH);
            pmiClnt.setStatLevel(nodeName, serverName, spec, true);

            Stats stats3 = pmiClnt.getStats(nodeName, serverName, oName, true);
            System.out.println("\n\n stats3 is");
            processStats(stats3);

            stats1.update(stats3, keepOld, recursiveUpdate);
            System.out.println("\n\n update stats1");
            processStats(stats1);

            stats2.update(stats3, keepOld, recursiveUpdate);
            System.out.println("\n\n update stats2");
            processStats(stats2);

        }
        catch(Exception ex) {
            System.out.println("\n\n Exception in testUpdate");
            ex.printStackTrace();
            success = false;
        }

    }


}
```

Retrieving performance data with PerfServlet

The PerfServlet is used for simple end-to-end retrieval of performance data that any tool, provided by either IBM or a third-party vendor, can handle.

Before you begin

The servlet provides a way to use an HTTP request to query the performance metrics for an entire WebSphere Application Server administrative domain. Because the servlet provides the performance data through HTTP, issues such as firewalls are trivial to resolve.

The PerfServlet provides the performance data output as an XML document, as described in the provided document type description (DTD). In the XML structure, the leaves of the structure provide the actual observations of performance data and the paths to the leaves that provide the context.

Starting with version 6.0, the PerfServlet in WebSphere Application Server uses the JMX Perf MBean interface to retrieve the PMI data and outputs an XML document that uses the Java Platform, Enterprise Edition (Java EE) 1.4 Performance Data Framework to describe the statistics. The PerfSerlvet in can also provide an output that is compatible with the PerfServlet 5.0. To provide PerfServlet 5.0 compatible output it uses the PMI client interface.

The performance servlet .ear file PerfServletApp.ear is located in the WAS_HOME/installableApps directory, where *WAS_HOME* is the installation path for WebSphere Application Server.

Starting with version 6.1, you must enable application security to get the PerfServlet working.

About this task

The performance servlet is deployed exactly as any other servlet. To use it, follow these steps:

1. Deploy the servlet on a single application server instance within the domain.
2. After the servlet deploys, you can invoke it to retrieve performance data for the entire domain. Invoke the performance servlet by accessing the following default URL:

   `http://hostname/wasPerfTool/servlet/perfservlet`

Results

The performance servlet provides performance data output as an XML document, as described by the provided document type definition (DTD). The DTD is located inside the PerfServletApp.ear file.

PerfServlet input

The PerfServlet input and output is used for simple end-to-end retrieval of performance data that any tool, provided by either IBM or a third-party vendor, can handle

The PerfServlet is deployed in one of the application server instance within the domain. By default, the PerfServlet collects all of the performance data across a WebSphere Application Server cell. However, it is possible to limit the data returned by the servlet to either a specific node, server, or PMI module:

Node .The servlet can limit the information it provides to a specific host by using the node parameter. For example, to limit the data collection to the node ‘rjones’, invoke the following URL:

```
http://hostname/wasPerfTool/servlet/perfservlet?node=rjones
```

Server

The servlet can limit the information it provides to a specific server by using the server parameter. For example, in order to limit the data collection to the ‘testserver’ server on all nodes, invoke the following URL:

```
http://hostname/wasPerfTool/servlet/perfservlet?server=testserver
```

To limit the data collection to the ‘testserver’ server located on the host ‘rjones’, invoke the following URL:

```
http://hostname/wasPerfTool/servlet/perfservlet?node=rjones&server=testserver
```

Module

The servlet can limit the information it provides to a specific PMI module by using the module parameter. You can request multiple modules by using the following URL:

```
http://hostname/wasPerfTool/servlet/perfservlet?module=beanModule+jvmRuntimeModule
```

For example, to limit the data collection to the beanModule on all servers and nodes, invoke the following URL:

```
http://hostname/wasPerfTool/servlet/perfservlet?module=beanModule
```

To limit the data collection to the beanModule on the server ‘testserver’ on the node rjones, invoke the following URL:

```
http://hostname/wasPerfTool/servlet/perfservlet?node=rjones&server=testserver&module=beanModule
```

To find the list of the modules, invoke the PerfServlet help with the following URL:

```
http://hostname/wasPerfTool/servlet/perfservlet?action=help
```

In WebSphere Application Server Network Deployment (ND), the PerfServlet automatically connects to the deployment manager to provide PMI data about the entire cell.

When the performance servlet is first initialized, it retrieves the list of nodes and servers within the domain in which it is deployed. Because the collection of this data is expensive, the performance servlet holds this

information as a cached list. If a new node is added to the domain or a new server is started, the performance servlet does not automatically retrieve the information about the newly created element. To force the servlet to refresh its configuration, you must add the refreshConfig parameter to the invocation as follows:

```
http://hostname/wasPerfTool/servlet/perfservlet?refreshConfig=true
```

You may want to configure other parameters of the performance servlets according to your specific needs. You can define the host, port number, connector type, and a user name and password.

- **Host**. This defines the host name where the server is running. The default value is ″localhost.″ For base installations, use ″localhost″ or ″host″ where application server is running.

  In Network Deployment installations, use the Deployment Manager's host name.

  Note: There can be more than one server instance in Network Deployment installations and a performance servlet can be installed in any one of the servers.
- **Port**. This is the port through which the server will connect. The default value is '8880' (SOAP connector port in base installation). In a base installation, use the application server SOAP or RMI connector port.

  In a Network Deployment installation, use the Deployment Manager's SOAP or RMI port.

  Note: The port numbers for SOAP/RMI connector can be configured in the administrative console under **Servers** > **Application Servers** > *server_name* > **End Points**.
- **Connector**. The connector type can be either SOAP or RMI. The default value is SOAP.
- **Username**. If security is enabled provide the user name.
- **Password**. If security is enabled provide the password.

PerfServlet output

The PerfServlet input and output is used for simple end-to-end retrieval of performance data that any tool, provided by either IBM or a third-party vendor, can handle.

The performance servlet .ear file PerfServletApp.ear is located in the `WAS_HOME/installableApps` directory.

The PerfServlet 6.0 provides output using the J2EE 1.4 Performance Data Framework. By default, the PerfServlet output is in 6.0 format. PerfServlet can provide the output in 5.0 format using the version parameter:

```
http://hostname/wasPerfTool/servlet/perfservlet?version=5
```

Refer to "PMI data classification" on page 7 for details about the Performance Data Framework.

PerfServlet 5.0 output details: The following section describes the PerfServlet 5.0 output. There are three types of leaves or output formats within the XML structure: PerfNumericInfo, PerfStatInfo, and PerfLoadInfo.

PerfNumericInfo:

When each invocation of the performance servlet retrieves the performance values from Performance Monitoring Infrastructure (PMI), some of the values are raw counters that record the number of times a specific event occurs during the lifetime of the server. If a performance observation is of the type PerfNumericInfo, the value represents the raw count of the number of times this event has occurred since the server started. This information is important to note because the analysis of a single document of data provided by the performance servlet might not be useful for determining the current load on the system. To determine the load during a specific interval of time, it might be necessary to apply simple statistical formulas to the data in two or more documents provided during this interval.

The PerfNumericInfo type has the following attributes:

time Specifies the time when the observation was collected (Java System.currentTimeMillis)

uid Specifies the PMI identifier for the observation

val Specifies the raw counter value

The following document fragment represents the number of loaded servlets. The path providing the context of the observation is not shown:

```
<numLoadedServlets>
     <PerfNumericData time="988162913175" uid="pmi1" val="132"/>
</numLoadedServlets>
```

PerfStatInfo:

When each invocation of the performance servlet retrieves the performance values from PMI, some of the values are stored as statistical data. Statistical data records the number of occurrences of a specific event, as the PerfNumericInfo type does. In addition, this type has sum of squares, mean, and total for each observation. This value is relative to when the server started.

The PerfStatInfo type has the following attributes:

time Specifies the time when the observation was collected (Java System.currentTimeMillis)

uid Specifies the PMI identifier for the observation

num Specifies the number of observations

sum_of_squares
Specifies the sum of the squares of the observations

total Specifies the sum of the observations

mean Specifies the mean (total number) for this counter

The following fragment represents the response time of an object. The path providing the context of the observation is not shown:

```
<responseTime>
     <PerfStatInfo mean="1211.5" num="5" sum_of_squares="3256265.0"
                   time="9917644193057" total="2423.0" uid="pmi13"/>
</responseTime>
```

PerfLoadInfo:

When each invocation of the performance servlet retrieves the performance values from PMI, some of the values are stored as a load. Loads record values as a function of time; they are averages. This value is relative to when the server started.

The PerfLoadInfo type has the following attributes:

time Specifies the time when the observation was collected (Java System.currentTimeMillis)

uid Specifies the PMI identifier for the observation

currentValue
Specifies the current value for this counter

integral
Specifies the time-weighted sum

timeSinceCreate
Specifies the elapsed time in milliseconds since this data was created in the server

mean Specifies time-weighted mean (integral/timeSinceCreate) for this counter

The following fragment represents the number of concurrent requests. The path providing the context of the observation is not shown:

```
<poolSize>
     <PerfLoadInfo currentValue="1.0" integral="534899.0" mean="0.9985028962051592"
                   time="991764193057" timeSinceCreate="535701.0" uid="pmi5"/>
</poolSize>
```

Related concepts

“PerfServlet input” on page 201
The PerfServlet input and output is used for simple end-to-end retrieval of performance data that any tool, provided by either IBM or a third-party vendor, can handle

Related tasks

“Retrieving performance data with PerfServlet” on page 200
The PerfServlet is used for simple end-to-end retrieval of performance data that any tool, provided by either IBM or a third-party vendor, can handle.

Using the JMX interface to develop your own monitoring application

You can use AdminClient API to get Performance Monitoring Infrastructure (PMI) data by using either PerfMBean or individual MBeans.

Before you begin

You can invoke methods on MBeans through the AdminClient Java Management Extension (JMX) interface. You can use AdminClient API to get Performance Monitoring Infrastructure (PMI) data by using either PerfMBean or individual MBeans. See information about using individual MBeans at bottom of this article.

Individual MBeans provide the Stats attribute from which you can get PMI data. The PerfMBean provides extended methods for PMI administration and more efficient ways to access PMI data. To set the PMI module instrumentation level, you must invoke methods on PerfMBean. To query PMI data from multiple MBeans, it is faster to invoke the getStatsArray method in PerfMBean than to get the Stats attribute from multiple individual MBeans. Perf MBean can provide PMI data from multiple MBeans using a single JMX call, but multiple JMX calls have to be made through individual MBeans.

See the topic ″Developing an administrative client program″ for more information on AdminClient JMX.

About this task

After the performance monitoring service is enabled and the application server is started or restarted, a PerfMBean is located in each application server giving access to PMI data. To use PerfMBean:

1. Create an instance of AdminClient. When using AdminClient API, you need to first create an instance of AdminClient by passing the host name, port number and connector type.

   The example code is:

```
AdminClient ac = null;
java.util.Properties props = new java.util.Properties();
props.put(AdminClient.CONNECTOR_TYPE, connector);
props.put(AdminClient.CONNECTOR_HOST, host);
props.put(AdminClient.CONNECTOR_PORT, port);
try {
    ac = AdminClientFactory.createAdminClient(props);
}
catch(Exception ex) {
    failed = true;
    new AdminException(ex).printStackTrace();
    System.out.println("getAdminClient: exception");
}
```

2. Use AdminClient to query the MBean ObjectNames Once you get the AdminClient instance, you can call queryNames to get a list of MBean ObjectNames depending on your query string. To get all the ObjectNames, you can use the following example code. If you have a specified query string, you will get a subset of ObjectNames.

```
javax.management.ObjectName on = new javax.management.ObjectName("WebSphere:*");
     Set objectNameSet= ac.queryNames(on, null);
 // you can check properties like type, name, and process to find a specified ObjectName
```

After you get all the ObjectNames, you can use the following example code to get all the node names:

```
  HashSet nodeSet = new HashSet();
        for(Iterator i = objectNameSet.iterator(); i.hasNext(); on =
(ObjectName)i.next()) {
            String type = on.getKeyProperty("type");
            if(type != null && type.equals("Server")) {
  nodeSet.add(servers[i].getKeyProperty("node"));
            }
 }
```

Note, this will only return nodes that are started. To list running servers on the node, you can either check the node name and type for all the ObjectNames or use the following example code:

```
     StringBuffer oNameQuery= new StringBuffer(41);
        oNameQuery.append("WebSphere:*");
        oNameQuery.append(",type=").append("Server");
        oNameQuery.append(",node=").append(mynode);

        oSet= ac.queryNames(new ObjectName(oNameQuery.toString()), null);
        Iterator i = objectNameSet.iterator ();
    while (i.hasNext ()) {
    on=(objectName) i.next();
   String process= on[i].getKeyProperty("process");
   serversArrayList.add(process);
  }
```

3. Get the PerfMBean ObjectName for the application server from which you want to get PMI data. Use this example code:

```
for(Iterator i = objectNameSet.iterator(); i.hasNext(); on = (ObjectName)i.next()) {
     // First make sure the node name and server name is what you want
     // Second, check if the type is Perf
            String type = on.getKeyProperty("type");
     String node = on.getKeyProperty("node");
     String process= on.getKeyProperty("process");
            if (type.equals("Perf") && node.equals(mynode) &
& server.equals(myserver)) {
  perfOName = on;
            }
 }
```

4. Invoke operations on PerfMBean through the AdminClient. Once you get the PerfMBean(s) in the application server from which you want to get PMI data, you can invoke the following operations on the PerfMBean through AdminClient API:

```
- setStatisticSet: Enable PMI data using the pre-defined statistic sets.
 Object[] params = new Object[] { com.ibm.websphere.pmi.stat.StatConstants.STATISTIC_SET_EXTENDED};
 String[] signature = new String[] {"java.lang.String"};
 ac.invoke (perfOName, "setStatisticSet", params, signature);

- getStatisticSet: Returns the current statistic set.
 String setname = (String) ac.invoke (perfOName, "getStatisticSet", null, null);

- setCustomSetString: Customizing PMI data that is enabled using fine-grained control.
This method allows to enable or disable statistics selectively. The format of the custom
set specification string is STATS_NAME=ID1,ID2,ID3 seperated by ':', where STATS_NAME
and IDs are defined in WS*Stat interfaces in com.ibm.websphere.pmi.stat package.
 params[0] = new String (WSJVMStats.NAME + "=" + WSJVMStats.HeapSize);
 params[1] = new Boolean (false);
```

```
 signature = new String[] {"java.lang.String", "java.lang.Boolean"};
 ac.invoke (perfOName, "setCustomSetString", params, signature);

  Note: Statistics that are not listed in the set string are not changed.

- getCustomSetString: Returns the current custom set specification as a string
 String setstring = (String) ac.invoke (perfOName, "getCustomSetString", null, null);


- setInstrumentationLevel: set the instrumentation level
            params[0] = new MBeanLevelSpec(objectName, new int[]{WSJVMStats.HEAPSIZE});
            params[1] = new Boolean(true);
            signature= new String[]{ "com.ibm.websphere.pmi.stat.MBeanLevelSpec",
"java.lang.Boolean"};
            ac.invoke(perfOName, "setInstrumentationLevel", params, signature);

- getInstrumentationLevel: get the instrumentation level
            params[0] = objectName;
            params[1] = new Boolean(recursive);
            String[] signature= new String[]{
"javax.management.ObjectName", "java.lang.Boolean"};
            MBeanLevelSpec[] mlss = (MBeanLevelSpec[])ac.invoke(perfOName,
     "getInstrumentationLevel", params, signature);

- setInstrumentationLevel: set the instrumentation level (deprecated in V6.0)
            params[0] = new MBeanLevelSpec(objectName, optionalSD, level);
            params[1] = new Boolean(true);
            signature= new String[]{ "com.ibm.websphere.pmi.stat.MBeanLevelSpec",
     "java.lang.Boolean"};
            ac.invoke(perfOName, "setInstrumentationLevel", params, signature);

- getInstrumentationLevel: get the instrumentation level (deprecated in V6.0)
            Object[] params = new Object[2];
            params[0] = new MBeanStatDescriptor(objectName, optionalSD);
            params[1] = new Boolean(recursive);
            String[] signature= new String[]{
     "com.ibm.websphere.pmi.stat.MBeanStatDescriptor", "java.lang.Boolean"};
            MBeanLevelSpec[] mlss = (MBeanLevelSpec[])ac.invoke(perfOName,
     "getInstrumentationLevel", params, signature);

- getConfigs: get PMI static config info for all the MBeans
            configs = (PmiModuleConfig[])ac.invoke(perfOName, "getConfigs", null, null);

- getConfig: get PMI static config info for a specific MBean
     ObjectName[] params = {objectName};
     String[] signature= { "javax.management.ObjectName" };
            config = (PmiModuleConfig)ac.invoke(perfOName, "getConfig", params,
signature);

- getStatsObject: you can use either ObjectName or MBeanStatDescriptor
            Object[] params    = new Object[2];
            params[0] = objectName;  // either ObjectName or or MBeanStatDescriptor
            params[1] = new Boolean(recursive);
            String[] signature = new String[] { "javax.management.ObjectName",
     "java.lang.Boolean"};
            Stats stats  = (Stats)ac.invoke(perfOName, "getStatsObject", params,
     signature);

  Note: The returned data only have dynamic information (value and time stamp).
See PmiJmxTest.java for additional code to link the configuration information with the
returned data.

- getStatsArray: you can use either ObjectName or MBeanStatDescriptor
            ObjectName[] onames = new ObjectName[]{objectName1, objectName2};
            Object[] params = new Object[]{onames, new Boolean(true)};
            String[] signature = new String[]{"[Ljavax.management.ObjectName;",
     "java.lang.Boolean"};
```

```
            Stats[] statsArray = (Stats[])ac.invoke(perfOName, "getStatsArray",
        params, signature);

      Note: The returned data only have dynamic information (value and time stamp).
    See PmiJmxTest.java for additional code to link the configuration information with the
    returned data.

    - listStatMembers: navigate the PMI module trees

            Object[] params = new Object[]{mName};
            String[] signature= new String[]{"javax.management.ObjectName"};
            MBeanStatDescriptor[] msds = (MBeanStatDescriptor[])ac.invoke(perfOName,
        "listStatMembers", params, signature);

    or,

            Object[] params = new Object[]{mbeanSD};
            String[] signature= new String[]
    {"com.ibm.websphere.pmi.stat.MBeanStatDescriptor"};
            MBeanStatDescriptor[] msds = (MBeanStatDescriptor[])ac.invoke
    (perfOName, "listStatMembers", params, signature);

    Refer the API documentation for deprecated classes
```

- **To use an individual MBean:** You need to get the AdminClient instance and the ObjectName for the individual MBean. Then you can simply get the Stats attribute on the MBean.

Example: Administering Java Management Extension-based interface

The page provides example code directly using Java Management Extension (JMX) API. For information on compiling your source code, see ″Compiling your monitoring applications.″

```
package com.ibm.websphere.pmi;

import com.ibm.websphere.management.AdminClient;
import com.ibm.websphere.management.AdminClientFactory;
import com.ibm.websphere.management.exception.ConnectorException;
import com.ibm.websphere.management.exception.InvalidAdminClientTypeException;
import com.ibm.websphere.management.exception.*;

import java.util.*;
import javax.management.*;
import com.ibm.websphere.pmi.*;
import com.ibm.websphere.pmi.client.*;
import com.ibm.websphere.pmi.stat.*;

/**
 * Sample code using AdminClient API to get PMI data from PerfMBean
 * and individual MBeans.
 *
 * @ibm-api
 */

public class PmiJmxTest implements PmiConstants
{

    private AdminClient    ac = null;
    private ObjectName     perfOName   = null;
    private ObjectName     serverOName = null;
    private ObjectName     wlmOName    = null;
    private ObjectName     jvmOName    = null;
    private ObjectName     orbtpOName    = null;
    private boolean failed = false;
    private PmiModuleConfig[] configs = null;

    /**
     *  Creates a new test object
     *  (Need a default constructor for the testing framework)
```

```
 */
public PmiJmxTest()
{
}

/**
 * @param args[0] host
 * @param args[1] port, optional, default is 8880
 * @param args[2] connectorType, optional, default is SOAP connector
 *
 */
public static void main(String[] args)
{
    PmiJmxTest instance = new PmiJmxTest();

    // parse arguments and create AdminClient object
    instance.init(args);

    // navigate all the MBean ObjectNames and cache those we are interested
    instance.getObjectNames();

    boolean v6 = !(new Boolean(System.getProperty ("websphereV5Statistics"))).
 booleanValue();

    if( v6 )
    {
        // test V6 APIs
        instance.doTestV6();
    }
    else
    {

        // set level, get data, display data
        instance.doTest();

        // test for EJB data
        instance.testEJB();

        // how to use JSR77 getStats method for individual MBean other than PerfMBean
        instance.testJSR77Stats();
    }

}

/**
 * parse args and getAdminClient
 */
public void init(String[] args)
{

    try
    {
        String  host    = null;
        String  port    = "8880";
        String  connector = AdminClient.CONNECTOR_TYPE_SOAP;
        if(args.length < 1) {
            System.err.println("ERROR: Usage: PmiJmxTest <host> [<port>] [<connector>]");
            System.exit(2);
        }
        else
        {
            host = args[0];

            if (args.length > 1)
                port = args[1];

            if (args.length > 2)
```

```
                connector = args[2];
            }

            if(host == null) {
                host = "localhost";
            }
            if(port == null) {
                port = "2809";
            }
            if (connector == null) {
                connector = AdminClient.CONNECTOR_TYPE_SOAP;
            }
            System.out.println("host=" + host + " , port=" + port + ",connector=" + connector);

            //-------------------------------------------------------------------
            // Get the ac object for the AppServer
            //-------------------------------------------------------------------
            System.out.println("main: create the adminclient");
            ac = getAdminClient(host, port, connector);

        }
        catch (Exception ex)
        {
            failed = true;
            new AdminException(ex).printStackTrace();
            ex.printStackTrace();
        }
    }

    /**
     * get AdminClient using the given host, port, and connector
     */
    public AdminClient getAdminClient(String hostStr, String portStr, String
    connector) {
        System.out.println("getAdminClient: host=" + hostStr + " , portStr=" + portStr);
        AdminClient ac = null;
        java.util.Properties props = new java.util.Properties();
        props.put(AdminClient.CONNECTOR_TYPE, connector);
        props.put(AdminClient.CONNECTOR_HOST, hostStr);
        props.put(AdminClient.CONNECTOR_PORT, portStr);

        /* set the following properties if security is enabled and using SOAP
    connector */
        /* The following shows how to set properties for SOAP connector when
    security is enabled.
           See AdminClient javadoc for more info.
        Properties props = new Properties();
       props.setProperty(AdminClient.CONNECTOR_HOST, "localhost");
       props.setProperty(AdminClient.CONNECTOR_PORT, "8880");
       props.setProperty(AdminClient.CONNECTOR_TYPE, AdminClient.CONNECTOR_
    TYPE_SOAP);
       props.setProperty(AdminClient.CONNECTOR_SECURITY_ENABLED, "true");
       props.setProperty(AdminClient.USERNAME, "test2");
      props.setProperty(AdminClient.PASSWORD, "user24test");
        props.setProperty("javax.net.ssl.trustStore",
    "C:/WebSphere/AppServer/etc/DummyClientTrustFile.jks");
     props.setProperty("javax.net.ssl.keyStore",
    "C:/WebSphere/AppServer/etc/DummyClientKeyFile.jks");
     props.setProperty("javax.net.ssl.trustStorePassword", "WebAS");
     props.setProperty("javax.net.ssl.keyStorePassword", "WebAS");
        */

        try {
            ac = AdminClientFactory.createAdminClient(props);
        }
        catch(Exception ex) {
            failed = true;
```

```
            new AdminException(ex).printStackTrace();
            System.out.println("getAdminClient: exception");
        }
        return ac;
    }


    /**
     * get all the ObjectNames.
     */
    public void getObjectNames() {

        try {

            //-----------------------------------------------------------------
            // Get a list of object names
            //-----------------------------------------------------------------
            javax.management.ObjectName on = new javax.management.ObjectName
("WebSphere:*");

            //-----------------------------------------------------------------
            // get all objectnames for this server
            //-----------------------------------------------------------------
           Set objectNameSet= ac.queryNames(on, null);

            //-----------------------------------------------------------------
            // get the object names that we care about: Perf, Server, JVM, WLM (only applicable in ND)
            //-----------------------------------------------------------------
            if(objectNameSet != null) {
                Iterator i = objectNameSet.iterator();
                while (i.hasNext()) {
                    on = (ObjectName)i.next();
                    String type = on.getKeyProperty("type");

                    // uncomment it if you want to print the ObjectName for each MBean
                    // System.out.println("\n\n" + on.toString());

                    // find the MBeans we are interested
                    if(type != null && type.equals("Perf")) {
                        System.out.println("\nMBean: perf =" + on.toString());
                        perfOName = on;
                    }
                    if(type != null && type.equals("Server")) {
                        System.out.println("\nMBean: Server =" + on.toString());
                        serverOName = on;
                    }
                    if(type != null && type.equals("JVM")) {
                        System.out.println("\nMBean: jvm =" + on.toString());
                        jvmOName = on;
                    }
                    if(type != null && type.equals("WLMAppServer")) {
                        System.out.println("\nmain: WLM =" + on.toString());
                        wlmOName = on;
                    }
                    if(type != null && type.equals("ThreadPool"))
                    {
                        String name = on.getKeyProperty("name");
                        if (name.equals("ORB.thread.pool"))
                            System.out.println("\nMBean: ORB ThreadPool =" + on.toString());
                        orbtpOName = on;
                    }
                }
            }
            else {
                System.err.println("main: ERROR: no object names found");
                System.exit(2);
            }
```

```
        // You must have Perf MBean in order to get PMI data.
        if (perfOName == null)
        {
            System.err.println("main: cannot get PerfMBean. Make sure PMI is enabled");
            System.exit(3);
        }
    }
    catch(Exception ex)
    {
        failed = true;
        new AdminException(ex).printStackTrace();
        ex.printStackTrace();
    }

}

/** Test V6 APIs */
public void doTestV6 ()
{
    System.out.println ("\ndoTestV6() output:\n");

    // the following methods are specific to V6 and demonstrates the V6 API..so set the flag to false
    String v5PropFlag = System.setProperty ("websphereV5Statistics", "false");
    try
    {
        Object[] params;
        String[] signature;

        // get current statistic set that is used for monitoring
        System.out.println ("\nCurrent statistic set: " + ac.invoke(perfOName, "getStatisticSet", null, null));


        // get all statistics from the server using Perf MBean
        System.out.println ("\nGet all statistics in PMI tree");
        signature = new String[]{"[Lcom.ibm.websphere.pmi.stat.StatDescriptor;","java.lang.Boolean"};
        params = new Object[] {new StatDescriptor[]{new StatDescriptor(null)}, new Boolean(true)};

        com.ibm.websphere.pmi.stat.WSStats[] wsStats = (com.ibm.websphere.pmi.stat.WSStats[])
        ac.invoke(perfOName, "getStatsArray", params, signature);
        System.out.println (wsStats[0].toString());


        // get statistics from one JVM MBean using J2EE JMX
        System.out.println ("\nGet JVM statistics using JVM MBean");
        javax.management.j2ee.statistics.Stats j2eeStats = (javax.management.j2ee.statistics.Stats)
        ac.getAttribute(jvmOName, "stats");
        System.out.println (j2eeStats.toString());


        // get statistics from a specific thread pool -- WebContainer thread pool
        System.out.println ("\nGet statistics for a specific thread pool");
        signature = new String[]{"[Lcom.ibm.websphere.pmi.stat.StatDescriptor;","java.lang.Boolean"};

        StatDescriptor webContainerPoolSD = new StatDescriptor (new String[] {WSThreadPoolStats.NAME,
 "WebContainer"});
        params = new Object[] {new StatDescriptor[]{webContainerPoolSD}, new Boolean(true)};

        wsStats = (com.ibm.websphere.pmi.stat.WSStats[])
        ac.invoke(perfOName, "getStatsArray", params, signature);
        System.out.println (wsStats[0].toString());


        // set monitoring to statistic set "extended"
        System.out.println ("\nSet monitoring to statistic set 'Extended'");
        signature = new String[]{"java.lang.String"};
        params = new Object[] {StatConstants.STATISTIC_SET_EXTENDED};
```

```
        ac.invoke(perfONName, "setStatisticSet", params, signature);

        // get current statistic set that is used for monitoring
        System.out.println ("\nCurrent statistic set: "+ ac.invoke(perfONName, "getStatisticSet", null, null));


        // selectively enable statistics for all thread pools
        System.out.println ("\nSelectively enable statistics (ActiveCount and PoolSize statistics)
   for thread pool -- fine grained control");

        StatDescriptor threadPoolSD = new StatDescriptor (new String[]
{WSThreadPoolStats.NAME});
            // create a spec object to enable ActiveCount and PoolSize on the thread pool
        StatLevelSpec[] spec = new StatLevelSpec[1];
        spec[0] = new StatLevelSpec (threadPoolSD.getPath(), new int[]
{WSThreadPoolStats.ActiveCount, WSThreadPoolStats.PoolSize});

        signature = new String[]{"[Lcom.ibm.websphere.pmi.stat.StatLevelSpec;","java.lang.Boolean"};
        params = new Object[] {spec, new Boolean(true)};

        ac.invoke(perfONName, "setInstrumentationLevel", params, signature);


        // get current statistic set that is used for monitoring
        System.out.println ("\nCurrent statistic set: "+ ac.invoke(perfONName, "getStatisticSet", null, null));

        // get statistics from all thread pools
        System.out.println ("\nGet statistics from all thread pools");
        signature = new String[]{"[Lcom.ibm.websphere.pmi.stat.StatDescriptor;","java.lang.Boolean"};
        params = new Object[] {new StatDescriptor[]{threadPoolSD},new Boolean(true)};

        wsStats = (com.ibm.websphere.pmi.stat.WSStats[])
        ac.invoke(perfONName, "getStatsArray", params, signature);
        System.out.println (wsStats[0].toString());

    }
    catch (Exception e)
    {
        e.printStackTrace();
    }

    // set the property to original value
    System.setProperty ("websphereV5Statistics", v5PropFlag);
}

/**
 * Some sample code to set level, get data, and display data. (V5)
 * @deprecated Use 6.0 APIs.
 */
public void doTest()
{
    try
    {
        // first get all the configs  - used to set static info for Stats
        // Note: server only returns the value and time info.
        //       No description, unit, etc is returned with PMI data to reduce communication cost.
        //       You have to call setConfig to bind the static info and Stats data later.
        configs = (PmiModuleConfig[])ac.invoke(perfONName, "getConfigs", null, null);

        // print out all the PMI modules and matching mbean types
        for (int i=0; i<configs.length; i++)
            System.out.println("config: moduleName=" + configs[i].getShortName() +
   ", mbeanType=" + configs[i].getMbeanType());

        // set the instrumentation level for the server
        setInstrumentationLevel(serverONName, null, PmiConstants.LEVEL_HIGH);
```

```
        // example to use StatDescriptor.
        // Note WLM module is only available in ND.
        StatDescriptor sd = new StatDescriptor(new String[] {"wlmModule.server"});
        setInstrumentationLevel(wlmOName, sd, PmiConstants.LEVEL_HIGH);

        // example to getInstrumentationLevel
        MBeanLevelSpec[] mlss = getInstrumentationLevel(wlmOName, sd, true);
        // you can call getLevel(), getObjectName(), getStatDescriptor() on mlss[i]

        // get data for the server
        Stats stats = getStatsObject(serverOName, true);
        System.out.println(stats.toString());

        // get data for WLM server submodule
        stats = getStatsObject(wlmOName, sd, true);
        if (stats == null)
            System.out.println("Cannot get Stats for WLM data");
        else
            System.out.println(stats.toString());

        // get data for JVM MBean
        stats = getStatsObject(jvmOName, true);
        processStats(stats);

        // get data for multiple MBeans
        ObjectName[] onames = new ObjectName[]{orbtpOName, jvmOName};
        Object[] params = new Object[]{onames, new Boolean(true)};
        String[] signature = new String[]{"[Ljavax.management.ObjectName;",
  "java.lang.Boolean"};
        Stats[] statsArray = (Stats[])ac.invoke(perfOName, "getStatsArray",
  params, signature);
        // you can call toString or processStats on statsArray[i]

        if (!failed)
            System.out.println("All tests passed");
        else
            System.out.println("Some tests failed");
    }
    catch(Exception ex)
    {
        new AdminException(ex).printStackTrace();
        ex.printStackTrace();
    }
}


/**
 * Sample code to get level
 */
protected MBeanLevelSpec[] getInstrumentationLevel(ObjectName on, StatDescriptor sd, boolean recursive)
{
    if (sd == null)
        return getInstrumentationLevel(on, recursive);
    System.out.println("\ntest getInstrumentationLevel\n");
    try {
        Object[] params = new Object[2];
        params[0] = new MBeanStatDescriptor(on, sd);
        params[1] = new Boolean(recursive);
        String[] signature= new String[]{ "com.ibm.websphere.pmi.stat.MBeanStatDescriptor", "java.lang.Boolean"};
        MBeanLevelSpec[] mlss = (MBeanLevelSpec[])ac.invoke(perfOName, "getInstrumentationLevel", params, signature)
        return mlss;
    }
    catch(Exception e) {
        new AdminException(e).printStackTrace();
        System.out.println("getInstrumentationLevel: Exception Thrown");
        return null;
```

```
    }
}

/**
 * Sample code to get level
 */
protected MBeanLevelSpec[] getInstrumentationLevel(ObjectName on,
 boolean recursive) {
    if (on == null)
        return null;
    System.out.println("\ntest getInstrumentationLevel\n");
    try {
        Object[] params = new Object[]{on, new Boolean(recursive)};
        String[] signature= new String[]{ "javax.management.ObjectName",
 "java.lang.Boolean"};
        MBeanLevelSpec[] mlss = (MBeanLevelSpec[])ac.invoke(perfOName,
 "getInstrumentationLevel", params, signature);
        return mlss;
    }
    catch(Exception e) {
        new AdminException(e).printStackTrace();
        failed = true;
        System.out.println("getInstrumentationLevel: Exception Thrown");
        return null;
    }
}

/**
 * Sample code to set level
 * @deprecated Use 6.0 APIs.
 */
protected void setInstrumentationLevel(ObjectName on, StatDescriptor sd,
 int level) {
    System.out.println("\ntest setInstrumentationLevel\n");
    try {
        Object[] params       = new Object[2];
        String[] signature    = null;
        MBeanLevelSpec[] mlss = null;
        params[0] = new MBeanLevelSpec(on, sd, level);
        params[1] = new Boolean(true);

        signature= new String[]{ "com.ibm.websphere.pmi.stat.MBeanLevelSpec","java.lang.Boolean"};
        ac.invoke(perfOName, "setInstrumentationLevel", params, signature);
    }
    catch(Exception e) {
        failed = true;
        new AdminException(e).printStackTrace();
        System.out.println("setInstrumentationLevel: FAILED: Exception Thrown");
    }
}

/**
 * Sample code to get a Stats object
 * @deprecated Use 6.0 APIs.
 */
public Stats getStatsObject(ObjectName on, StatDescriptor sd, boolean
 recursive) {

    if (sd == null)
        return getStatsObject(on, recursive);

    System.out.println("\ntest getStatsObject\n");
    try {
        Object[] params    = new Object[2];
        params[0] = new MBeanStatDescriptor(on, sd);  // construct MBeanStatDescriptor
        params[1] = new Boolean(recursive);
        String[] signature = new String[] { "com.ibm.websphere.pmi.stat.MBeanStatDescriptor", "java.lang.Boolean"};
```

```
        Stats stats  = (Stats)ac.invoke(perfONName, "getStatsObject", params, signature);

        if (stats == null) return null;


        // find the PmiModuleConfig and bind it with the data
        String type = on.getKeyProperty("type");
        if (type.equals(MBeanTypeList.SERVER_MBEAN))
            setServerConfig(stats);
        else
            stats.setConfig(PmiClient.findConfig(configs, on));

        return stats;

    } catch(Exception e) {
        failed = true;
        new AdminException(e).printStackTrace();
        System.out.println("getStatsObject: Exception Thrown");
        return null;
    }
}

/**
 * Sample code to get a Stats object
 */
public Stats getStatsObject(ObjectName on, boolean recursive) {
    if (on == null)
        return null;

    System.out.println("\ntest getStatsObject\n");

    try {
        Object[] params  = new Object[]{on, new Boolean(recursive)};
        String[] signature = new String[] { "javax.management.ObjectName",
 "java.lang.Boolean"};
        Stats stats  = (Stats)ac.invoke(perfONName, "getStatsObject", params,
 signature);


        // find the PmiModuleConfig and bind it with the data
        String type = on.getKeyProperty("type");
        if (type.equals(MBeanTypeList.SERVER_MBEAN))
            setServerConfig(stats);
        else
            stats.setConfig(PmiClient.findConfig(configs, on));

        return stats;

    }
    catch(Exception e) {
        failed = true;
        new AdminException(e).printStackTrace();
        System.out.println("getStatsObject: Exception Thrown");
        return null;
    }
}

/**
 * Sample code to navigate and get the data value from the Stats object.
 */
private void processStats(Stats stat) {
    processStats(stat, "");
}

/**
  * Sample code to navigate and get the data value from the Stats and
Statistic object.
```

```
     * @deprecated Use 6.0 APIs.
     */
    private void processStats(Stats stat, String indent) {
        if(stat == null)  return;

        System.out.println("\n\n");

        // get name of the Stats
        String name = stat.getName();
        System.out.println(indent + "stats name=" + name);

        // list data names
        String[] dataNames = stat.getStatisticNames();
        for (int i=0; i<dataNames.length; i++)
            System.out.println(indent + "    " + "data name=" + dataNames[i]);
        System.out.println("");

        // list all datas
        //com.ibm.websphere.management.statistics.Statistic[] allData =
    stat.getStatistics();

        // cast it to be PMI's Statistic type so that we can have get more
        // Also show how to do translation.
        //Statistic[] dataMembers = (Statistic[])allData;
        Statistic[] dataMembers = stat.listStatistics();
        if(dataMembers != null) {
            for(int i=0; i<dataMembers.length; i++)  {
                System.out.print(indent + "    " + "data name=" +
    PmiClient.getNLSValue(dataMembers[i].getName())
                                + ", description=" + PmiClient.getNLSValue
    (dataMembers[i].getDescription())
                                + ", startTime=" + dataMembers[i].getStartTime()
                                + ", lastSampleTime=" + dataMembers[i].getLastSampleTime());
                if(dataMembers[i].getDataInfo().getType() == TYPE_LONG) {
                    System.out.println(", count=" + ((CountStatisticImpl)dataMembers[i]).getCount());
                }
                else if(dataMembers[i].getDataInfo().getType() == TYPE_STAT) {
                    TimeStatisticImpl data = (TimeStatisticImpl)dataMembers[i];
                    System.out.println(", count=" + data.getCount()
                                       + ", total=" + data.getTotal()
                                       + ", mean=" + data.getMean()
                                       + ", min=" + data.getMin()
                                       + ", max=" + data.getMax());
                }
                else if(dataMembers[i].getDataInfo().getType() == TYPE_LOAD) {
                    RangeStatisticImpl data = (RangeStatisticImpl)dataMembers[i];
                    System.out.println(", current=" + data.getCurrent()
                                       + ", integral=" + data.getIntegral()
                                       + ", avg=" + data.getMean()
                                       + ", lowWaterMark=" + data.getLowWaterMark()
                                       + ", highWaterMark=" + data.getHighWaterMark());
                }
            }
        }

        // recursively for sub-stats
        Stats[] substats = (Stats[])stat.getSubStats();
        if(substats == null || substats.length == 0)
            return;
        for(int i=0; i<substats.length; i++) {
            processStats(substats[i], indent + "    ");
        }
    }

    /**
     * The Stats object returned from server does not have static config info.
    You have to set it on client side.
```

```
 */
public void setServerConfig(Stats stats) {
    if(stats == null) return;
    if(stats.getType() != TYPE_SERVER) return;

    PmiModuleConfig config = null;

    Stats[] statList = stats.getSubStats();
    if (statList == null || statList.length == 0)
        return;
    Stats oneStat = null;
    for(int i=0; i<statList.length; i++) {
        oneStat = statList[i];
        if (oneStat == null) continue;
        config = PmiClient.findConfig(configs, oneStat.getStatsType());
 //getName
        if(config != null)
            oneStat.setConfig(config);
        else
        {
            config = getStatsConfig (oneStat.getStatsType());
            if (config != null)
                oneStat.setConfig(config);
            else
                System.out.println("Error: get null config for " + oneStat.getStatsType());
        }
    }
}

/**
 * sample code to show how to get a specific MBeanStatDescriptor
 * @deprecated Use 6.0 APIs.
 */
public MBeanStatDescriptor getStatDescriptor(ObjectName oName, String name) {
    try {
        Object[] params = new Object[]{serverOName};
        String[] signature= new String[]{"javax.management.ObjectName"};
        MBeanStatDescriptor[] msds = (MBeanStatDescriptor[])ac.invoke
(perfOName, "listStatMembers", params, signature);
        if (msds == null)
            return null;
        for (int i=0; i<msds.length; i++) {
            if (msds[i].getName().equals(name))
                return msds[i];
        }
        return null;
    }
    catch(Exception e) {
        new AdminException(e).printStackTrace();
        System.out.println("listStatMembers: Exception Thrown");
        return null;
    }

}

/**
 * sample code to show you how to navigate MBeanStatDescriptor via
listStatMembers
 * @deprecated Use 6.0 APIs.
 */
public MBeanStatDescriptor[] listStatMembers(ObjectName mName) {
    if (mName == null)
        return null;

    try {
        Object[] params = new Object[]{mName};
        String[] signature= new String[]{"javax.management.ObjectName"};
```

```
        MBeanStatDescriptor[] msds = (MBeanStatDescriptor[])ac.invoke
 (perfOName, "listStatMembers", params, signature);
        if (msds == null)
            return null;
        for (int i=0; i<msds.length; i++) {
            MBeanStatDescriptor[] msds2 = listStatMembers(msds[i]);
        }
        return null;
    }
    catch(Exception e) {
        new AdminException(e).printStackTrace();
        System.out.println("listStatMembers: Exception Thrown");
        return null;
    }

}


/**
 * Sample code to get MBeanStatDescriptors
 * @deprecated Use 6.0 APIs.
 */
public MBeanStatDescriptor[] listStatMembers(MBeanStatDescriptor mName) {
    if (mName == null)
        return null;

    try {
        Object[] params = new Object[]{mName};
        String[] signature= new String[]{"com.ibm.websphere.pmi.stat.MBeanStatDescriptor"};
        MBeanStatDescriptor[] msds = (MBeanStatDescriptor[])ac.invoke(perfOName, "listStatMembers",
  params, signature);
        if (msds == null)
            return null;
        for (int i=0; i<msds.length; i++) {
            MBeanStatDescriptor[] msds2 = listStatMembers(msds[i]);
            // you may recursively call listStatMembers until find the one you want
        }
        return msds;
    }
    catch(Exception e) {
        new AdminException(e).printStackTrace();
        System.out.println("listStatMembers: Exception Thrown");
        return null;
    }

}

/**
 * sample code to get PMI data from beanModule
 * @deprecated Use 6.0 APIs.
 */
public void testEJB() {

    // This is the MBeanStatDescriptor for Enterprise EJB
    MBeanStatDescriptor beanMsd = getStatDescriptor(serverOName,
 PmiConstants.BEAN_MODULE);
    if (beanMsd == null)
        System.out.println("Error: cannot find beanModule");

    // get the Stats for module level only since recursive is false
    Stats stats = getStatsObject(beanMsd.getObjectName(), beanMsd.
getStatDescriptor(), false); // pass true if you wannt data from individual beans

    // find the avg method RT
    TimeStatisticImpl rt = (TimeStatisticImpl)stats.getStatistic
(EJBStatsImpl.METHOD_RT);
    System.out.println("rt is " + rt.getMean());
```

```
    try {
        java.lang.Thread.sleep(5000);
    } catch (Exception ex) {
        ex.printStackTrace();
    }

    // get the  Stats again
    Stats stats2 = getStatsObject(beanMsd.getObjectName(), beanMsd.
getStatDescriptor(), false); // pass true if you wannt data from individual beans

    // find the avg method RT
    TimeStatisticImpl rt2 = (TimeStatisticImpl)stats2.getStatistic
(EJBStatsImpl.METHOD_RT);
    System.out.println("rt2 is " + rt2.getMean());

    // calculate the difference between this time and last time.
    TimeStatisticImpl deltaRt = (TimeStatisticImpl)rt2.delta(rt);
    System.out.println("deltaRt is " + rt.getMean());

}

/**
 * Sample code to show how to call getStats on StatisticProvider MBean
directly.
 * @deprecated Use 6.0 APIs.
 */
public void testJSR77Stats() {
    // first, find the MBean ObjectName you are interested.
    // Refer method getObjectNames for sample code.

    // assume we want to call getStats on JVM MBean to get statistics
    try {

        com.ibm.websphere.management.statistics.JVMStats stats =
                (com.ibm.websphere.management.statistics.JVMStats)ac.
invoke(jvmONname, "getStats", null, null);

        System.out.println("\n get data from JVM MBean");

        if (stats == null) {
            System.out.println("WARNING: getStats on JVM MBean returns null");
        } else {

            // first, link with the static info if you care
            ((Stats)stats).setConfig(PmiClient.findConfig(configs, jvmONname));

            // print out all the data if you want
            //System.out.println(stats.toString());

            // navigate and get the data in the stats object
            processStats((Stats)stats);

            // call JSR77 methods on JVMStats to get the related data
            com.ibm.websphere.management.statistics.CountStatistic upTime =
  stats.getUpTime();
            com.ibm.websphere.management.statistics.BoundedRangeStatistic
  heapSize = stats.getHeapSize();

            if (upTime != null)
                System.out.println("\nJVM up time is " + upTime.getCount());
            if (heapSize != null)
                System.out.println("\nheapSize is " + heapSize.getCurrent());
        }
    } catch (Exception ex) {
        ex.printStackTrace();
        new AdminException(ex).printStackTrace();
```

```
        }
    }


    /**
     Get PmiModuleConfig from server
     */
    public PmiModuleConfig getStatsConfig (String statsType)
    {
       try
       {
            return (PmiModuleConfig)ac.invoke(perfOName, "getConfig",
                                              new String[]{statsType},
                                              new String[]{"java.lang.String"});
       }
       catch(Exception e)
       {
            e.printStackTrace();
            return null;
       }
    }

    /**
     * Get PmiModuleConfig based on MBean ObjectName
      @deprecated Use com.ibm.websphere.pmi.client.PmiClient.findConfig()
     */
    public PmiModuleConfig findConfig(ObjectName on) {
        if (on == null) return null;

        String type = on.getKeyProperty("type");
        System.out.println("findConfig: mbean type =" + type);

        for (int i=0; i<configs.length ; i++) {

            if (configs[i].getMbeanType().equals(type))
                return configs[i];
        }
        System.out.println("Error: cannot find the config");
        return null;

    }

    /**
     * Get PmiModuleConfig based on PMI module name
     @deprecated Use com.ibm.websphere.pmi.client.PmiClient.findConfig()
     */
    public PmiModuleConfig findConfig(String moduleName) {
        if (moduleName == null) return null;

        for (int i=0; i<configs.length ; i++) {

            if (configs[i].getShortName().equals(moduleName))
                return configs[i];
        }
        System.out.println("Error: cannot find the config");
        return null;

    }
}
```

Developing PMI interfaces (Version 4.0) (deprecated)

You can use the Performance Monitoring Infrastructure (PMI) interfaces to develop your own applications to collect and display performance information.

About this task

This section discusses the use of the Performance Monitoring Infrastructure (PMI) client interfaces in applications. Read the basic steps in the programming model:

1. Retrieve an initial collection or snapshot of performance data from the server. A client uses the CpdCollection interface to retrieve an initial collection or snapshot from the server. This snapshot, which is called Snapshot in this example, is provided in a hierarchical structure as described in data organization and hierarchy, and contains the current values of all performance data collected by the server. The snapshot maintains the same structure throughout the lifetime of the CpdCollection instance.
2. Process and display the data as specified. The client processes and displays the data as specified. Processing and display objects, for example, filters and GUIs, can register as CpdEvent listeners to data of interest. The listener works only within the same Java virtual machine (JVM). When the client receives updated data, all listeners are notified.
3. Display the new CpdCollection instance through the hierarchy. When the client receives new or changed data, the client can simply display the new CpdCollection instance through its hierarchy. When it is necessary to update the Snapshot collection, the client can use the update method to update Snapshot with the new data.

   ```
   Snapshot.update(S1);
   // ...later...
   Snapshot.update(S2);
   ```

Results

Steps 2 and 3 are repeated through the lifetime of the client.

Compiling your monitoring applications

Use this page to find the JAR files required to compile your Performance Monitoring Infrastructure (PMI).

To compile your Performance Monitoring Infrastructure (PMI) code, you must have the following JAR file in your class path:

- %WAS_HOME%/plugins/com.ibm.ws.runtime_6.1.0.jar

What to do next

The JAR files listed above are needed to compile the PmiJmxTest example application. If your monitoring applications use APIs in other packages, also include those packages on the class path. If any WebSphere Application Server class is not found with the above set of jars, then you can include all the WebSphere jars using:

```
javac -classpath %WAS_HOME%\plugins\com.ibm.ws.runtime_6.1.0.jar myclass.java
```

Running your new monitoring applications

Use this page to learn about the steps you must follow in order to run monitoring applications.

About this task

Follow these steps to run your monitoring applications.

1. You need a WebSphere Application Server installation or WebSphere Application Server Java Platform, Enterprise Edition (Java EE) client package to run a PMI application.
2. Use a PMI client API to write your own application.
3. Compile the newly-written PMI application and place it on the class path. (The jar files under %WAS_HOME%\lib and %WAS_HOME%\classes folder will be placed in the class path by the following script.)

4. To run a PMI application you need a WebSphere Application Server runtime environment (the application server installation or a Java EE client package). Using the following script to run the application:

   Note: The following is formatted for Windows® based systems. You may need to adjust the script depending on your operating system.

```
@echo off
@setlocal

call "%~dp0setupCmdLine.bat"

"%JAVA_HOME%\bin\java" "%CLIENTSAS%" "%CLIENTSOAP%" -DwebsphereV5Statistics=false
-Dwas.install.root="%WAS_HOME%" -Dws.ext.dirs="%WAS_EXT_DIRS%" -classpath "%WAS_CLASSPATH%"
com.ibm.ws.bootstrap.WSLauncher com.ibm.websphere.pmi.PmiJmxTest %*
```

Performance Monitoring Infrastructure client package

Use this page to learn how to use the PmiClient application and JMX connector to communicate to the Perf MBean in an application server.

A Performance Monitoring Infrastructure (PMI) client package provides a PmiClient wrapper class to deliver PMI data to a client. As shown in the following figure, the PmiClient API uses the AdminClient API to communicate to the Perf MBean in an application server.

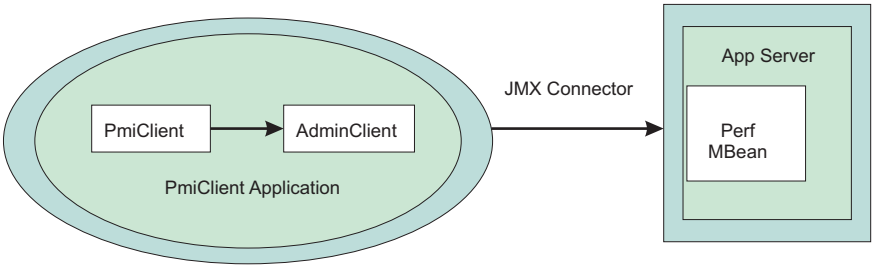
The PmiClient communicates with the network manager first, retrieving an AdminClient instance to each application server. When the PmiClient receives the instance, it uses it to communicate with the application server directly for performance or level setting changes. Since level settings are persistent through PmiClient, you are only required to set it once, unless you want to change it.

Performance Monitoring Infrastructure and Java Management Extensions

The PmiClient API does not work if the Java Management Extensions (JMX) infrastructure and Perf MBean are not running. If you prefer to use the AdminClient API directly to retrieve PMI data, you still have a dependency on the JMX infrastructure.

When using the PmiClient API, you have to pass the JMX connector protocol and port number to instantiate an object of the PmiClient. Once you get a PmiClient object, you can call its methods to list nodes, servers and MBeans, set the monitoring level, and retrieve PMI data.

The PmiClient API creates an instance of the AdminClient API and delegates your requests to the AdminClient API. The AdminClient API uses the JMX connector to communicate with the Perf MBean in the corresponding server and then returns the data to the PmiClient, which returns the data to the client.



Running your monitoring applications with security enabled

You can opt to run a Performance Monitoring Infrastructure client application with security enabled.

About this task

In order to run a Performance Monitoring Infrastructure (PMI) client application with security enabled, you must have %CLIENTSOAP% properties defined on your Java virtual machine command line for a SOAP connector, and %CLIENTSAS% properties defined for a RMI connector. The %CLIENTSOAP% and %CLIENTSAS% properties are defined in the setupCmdLine.bat or setupCmdline.sh files.

- If you are configuring security for a SOAP connector:
  1. Set com.ibm.SOAP.securityEnabled to **True** in the soap.client.props file for the SOAP connector. The soap.client.props property file is located in the *profile_root*/properties directory.
  2. Set com.ibm.SOAP.loginUserid and com.ibm.SOAP.loginPassword as the user ID and password for login.
- If you are configuring security for a RMI connector, set the sas.client.props file or type the user ID and password in the window, if you do not put them in the property file for Remote Method Invocation (RMI) connector. A common mistake is to leave extra spaces at the end of the lines in the property file. Do not leave extra spaces at the end of the lines, especially for the user ID and password lines.

Monitoring performance with Tivoli Performance Viewer (TPV)

Tivoli Performance Viewer (TPV) enables administrators and programmers to monitor the overall health of WebSphere Application Server from within the administrative console.

Before you begin

In Version 4.0, Tivoli Performance Viewer was named the Resource Analyzer. In Version 5.0, TPV is a standalone Java application. In Versions 6.0.x and later, TPV is embedded in the administrative console. From TPV, you can view current activity and summary reports, or log Performance Monitoring Infrastructure (PMI) performance data. TPV provides a simple viewer for the performance data collected by the Performance Monitoring Infrastructure.

About this task

By viewing TPV data, administrators can determine which part of the application and configuration settings to change in order to improve performance. For example, you can view the servlet summary reports, enterprise beans, and Enterprise JavaBeans (EJB) methods in order to determine what part of the application to focus on. Then, you can sort these tables to determine which of these resources has the highest response time. Focus on improving the configuration for those application resources taking the longest response time.

If a problem happens when using Tivoli Performance Viewer to view Performance Monitoring Infrastructure (PMI) data, refer to the troubleshooting Technote.

1. Optional: Adjust the Performance Monitoring Infrastructure (PMI) settings for the servers that you want to monitor. The PMI service is enabled by default with a basic set of counters enabled.
2. Monitor current server activity. You can view real-time data on the current performance activity of a server using TPV in the administrative console.
   - Use the performance advisors to examine various data while your application is running. The performance advisor in TPV provides advice to help tune systems for optimal performance by using collected PMI data.
   - Configure user and logging settings for TPV. These settings may affect the performance of your application server.
   - View summary reports on servlets, Enterprise JavaBeans (EJB) methods, connections pools and thread pools in WebSphere Application Server.
   - View performance modules that provide graphs and of various performance data on system resources such as CPU utilization, on WebSphere pools and queues such as database connection

pools, and on customer application data such as servlet response time. In addition to providing a viewer for performance data, TPV enables you to view data for other products or customer applications that have implemented custom PMI.

3. View server performance logs. You can record and view data that has been logged by TPV in the administrative console. Be sure to configure user and logging settings for TPV.
4. Log performance data. You can record real-time data for later retrieval and analysis.

Why use Tivoli Performance Viewer?

Administrators and programmers can monitor the overall health of WebSphere Application Server from within the administrative console by using Tivoli Performance Viewer (TPV).

From TPV, you can view current activity or log Performance Monitoring Infrastructure (PMI) performance data for the following:

- System resources such as CPU utilization
- WebSphere pools and queues such as a database connection pool
- Customer application data such as average servlet response time

You can also view data for other products or customer applications that implement custom PMI by using TPV. For more information on custom PMI, refer to “Enabling PMI data collection” on page 174.

By viewing PMI data, administrators can determine which part of the application and configuration settings to alter in order to improve performance. For example, in order to determine what part of the application to focus on, you can view the servlet summary report, enterprise beans and Enterprise JavaBeans (EJB) methods, and determine which of these resources has the highest response time. You can then focus on improving the configuration for those application resources with the longest response times.

Tivoli Performance Viewer is used to help manage configuration settings by viewing the various graphs or using the Tivoli Performance Advisor. For example, by looking at the summary chart for thread pools, you can determine whether the thread pool size needs to be increased or decreased by monitoring the percent (%) usage. After configuration settings are changed based on the data provided, you can determine the effectiveness of the changes. To help with configuration settings, use the Tivoli Performance Advisor. The Advisor assesses various data while your application is running, and provides configuration setting advice to improve performance.

Note: In Version 4.0, Tivoli Performance Viewer was originally named the Resource Analyzer. In Version 5.0, TPV was a standalone Java application. In Versions 6.0.*x* and later, TPV is embedded in the administrative console.

TPV topologies and performance impacts

The Tivoli Performance Viewer (TPV) enables administrators and programmers to monitor the overall health of WebSphere Application Server from within the administrative console.

The following two topologies exist for the Tivoli Performance Viewer:

- Tivoli Performance Viewer running in a single server environment
- Tivoli Performance Viewer running in a Network Deployment environment

When the Tivoli Performance Viewer is running in a single server environment, the collection and viewing of the data occurs in the same Java virtual machine. Because the collection and viewing of data occurs in the application server, performance may be affected depending on TPV settings. “Viewing Data with the Tivoli Performance Viewer” on page 228 describes the various TPV settings and their effect on performance.

When the Tivoli Performance Viewer is used in a Network Deployment environment, the data is collected at each of the nodes and stored in memory at the administrative agent. Data is then viewed from the

deployment manager. This architecture enables the monitoring work to be distributed among the nodes. Similar to the single server environment, the various user and logging settings directly influence the extent to which performance is affected.

Related tasks

"Monitoring performance with Tivoli Performance Viewer (TPV)" on page 223
Tivoli Performance Viewer (TPV) enables administrators and programmers to monitor the overall health of WebSphere Application Server from within the administrative console.

Viewing current performance activity

You can view the current performance activity of a server using the Tivoli Performance Viewer (TPV) in the administrative console.

Before you begin

Once monitoring is enabled, TPV monitors the performance activity of all servers on a node, which includes the associated application servers and the administrative agent for the node being monitored.

About this task

TPV enables administrators and programmers to monitor the current health of WebSphere Application Server. Because the collection and viewing of data occurs in the application server, performance may be affected. To minimize performance impacts, monitor only those servers whose resources need to be optimized.

1. Click **Monitoring and Tuning > Performance Viewer > Current Activity** in the console navigation tree. The TPV current activity collection is displayed.
2. Start monitoring the current activity of a server in either of two ways:
   - Under **Server**, click the name of the server whose activity you want to monitor. Clicking on the name starts the monitoring for the server and displays the activity page for the server.
   - Select the check box for the server whose activity you want to monitor, and click **Start Monitoring**.To start monitoring multiple servers at the same time, select the servers and click **Start Monitoring**.

   A TPV console panel is displayed, providing a navigation tree on the left and a view of real-time data on the current performance activity of a server on the right.
3. From the navigation tree, select the server activity data that you want to view.

Option	Description
Advisor	Use the Performance Advisor to examine various data while your application is running. The Performance Advisor provides advice to help tune systems for optimal performance and gives recommendations on inefficient settings by using collected PMI data.
Settings	Configure user and logging settings for TPV. These settings can affect the performance of your application server.
Summary Reports	View summary reports on servlets, enterprise beans (EJBs), EJB methods, connections pools and thread pools in WebSphere Application Server.

Option	Description
Performance Modules	View performance modules that provide graphics and charts of various performance data on system resources such as CPU utilization, on WebSphere pools and queues such as database connection pools, and on customer application data such as servlet response time. In addition to providing a viewer for performance data, TPV enables you to view data for other products or customer applications that have implemented custom PMI.

What to do next

When you finish monitoring a server, select the server and click **Stop Monitoring**. TPV automatically stops monitoring a server if you the browser window is closed or if the user logs out.

Selecting a server and changing monitoring status

Use this page to start and stop monitoring for each server and to select a server for Tivoli Performance Viewer. You can also view the collection status for each server.

On this page, you can view the collection status of each server. When the collection status is **Monitored**, Tivoli Performance Monitor is enabled. If the collection status is **Available**, the server is ready to be enabled. If the collection status is **Unavailable, server stopped, PMI not enabled, or nodeagent stopped**, you must restart your server before you can start monitoring. Click on any server or node agent to view the current activity for that server. Lastly, **Performance Data Collection Not Supported** means that the server is not version 6.0.x or later.

Maximum rows: Specifies the maximum number of rows that displays when the collection is large. The rows that are not displayed appear on the next page.

Retain filter criteria: Specifies whether to use the same filter criteria entered in the show filter function to display this page the next time you visit it.

Start monitoring:

Select one or more servers from the list and press Start monitoring to start the Tivoli Performance Monitor for the selected servers.

Stop monitoring:

Select one or more servers from the list and press Stop monitoring to stop the Tivoli Performance Monitor for the selected servers.

Configuring TPV settings

You can configure user and logging settings of the Tivoli Performance Viewer (TPV). Configuring the TPV settings affects the performance of your application server.

About this task

TPV monitors the performance activity of all servers on a node.

The data can also be viewed from the deployment manager.

You can configure the activity monitoring of TPV on a per-user basis. Any changes made to TPV settings are only for the server being monitored and only affect the user viewing the data.

You can change the user and log TPV settings in the administrative console.

- Configure the TPV user settings.
  1. Click **Monitoring and Tuning > Performance Viewer > Current Activity >** *server_name* **> Settings > User** in the console navigation tree. To see the **User** link on the Tivoli Performance Viewer page, expand the **Settings** node of the TPV navigation tree on the left side of the page. After clicking **User**, the TPV user settings are displayed on the right side of the page.
  2. Change the values as needed for the user settings. The settings are described briefly below and in more detail in the Tivoli Performance Viewer settings.

Refresh Rate	Specifies how frequently TPV collects performance data for a server from the Performance Monitoring Infrastructure (PMI) service provided by that server. The default is 30 seconds. To collect performance data for the server more frequently, set the refresh rate to a smaller number. To collect performance data less frequently, set the refresh rate to a larger number. The allowed range is 5 to 500 seconds.
Buffer Size	Specifies the number of entries to be stored for a server. Data displayed in TPV is stored in a short in-memory buffer. After the buffer is full, each time a new entry is retrieved the oldest entry is discarded. The default buffer size is 40 entries. Supported values are 10, 20, 30, 40, 50, 60, 70, 80, 90 and 100. The larger the buffer size, the more memory is consumed. Thus, specify a buffer size that allows you capture enough monitoring data for analysis without wasting memory storing unneeded data.
View Data As	Specifies how counter values are displayed. Viewing options include the following: **Raw Value** Displays the absolute value. If the counter represents load data, such as the average number of connections in a database pool, then TPV displays the current value followed by the average. For example, `18 (avg:5)`. **Change in Value** Displays the change in the current value from the previous value. **Rate of Change** Displays the ratio `change/(T1 - T2)`, where *change* is the change in the current value from the previous value, *T1* is the time when the current value was retrieved, and *T2* is the time when the previous value was retrieved.

     The refresh rate and buffer size settings combine to control how much temporal history you have for the application server. The default values for **Refresh Rate** (30 seconds) and **Buffer Size** (40 entries) provide you with a 20-minute history of the application server's performance. Changing one of these parameters affects the length of the temporal history.

     The values you set for **Refresh Rate** and **Buffer Size** depend on your use of TPV. To diagnose a known problem on a test machine, you might poll data more frequently while having a decreased buffer size. To monitor a production server, you might poll data less frequently and specify a buffer size depending on how much history you want. However, TPV is not intended to be a full-time monitoring solution.
  3. Click **Apply**.

- Configure the TPV log settings. The log settings control what happens when **Start Logging** is clicked in, for example, a summary report on the performance of a servlet, enterprise bean (EJB), EJB method, connection pool or thread pool.
  1. Click **Monitoring and Tuning > Performance Viewer > Current Activity >** *server_name* **> Settings > Log** in the console navigation tree. To see the **Log** link on the Tivoli Performance Viewer page, expand the **Settings** node of the TPV navigation tree on the left side of the page. After clicking **Log**, the TPV log settings are displayed on the right side of the page.
  2. Change the values as needed for the log settings. The settings are described below and in the Tivoli Performance Viewer settings.

Duration	Specifies the length of time, in minutes, that logging continues, unless **Stop Logging** is clicked first. TPV is not intended as a full-time logging solution.
Maximum File Size	Specifies the maximum size, in megabytes, of a single file. Note that TPV automatically zips log files to save space and this parameter controls the pre-zipped file size and not the post-zipped, which is smaller.
Maximum Number of Historical Files	Specifies the number of files TPV writes before stopping. If TPV reaches the maximum file size before the logging duration ends, it continues logging in another file, up to the maximum. If TPV reaches the maximum number of historical files before the logging duration ends, TPV deletes the oldest historical file and continues logging in a new file. The total amount of data that is stored is constrained by the **Maximum File Size** and **Maximum Number of Historical Files** parameters.
File Name	Specifies the name of the log file. The server name and the time at which the log is started is appended to the log name to help users identify a log file.
Log Output Format	Specifies whether TPV writes log files as XML or in a binary format. Binary format is recommended as it provides a smaller log file when uncompressed.

  3. Click **Apply**.

Viewing Data with the Tivoli Performance Viewer

Use this page to view and refresh performance data for the selected server, change user and log settings, view summary reports, and information on specific performance modules.

To view this administrative console page, click **Monitoring and Tuning** > **Performance Viewer** > **Current Activity** > *server*.

Click the server name to view the current activity for that server. In this view, the Tivoli Performance Viewer (TPV) has two main parts, which include the navigation panel located beside the administrative console navigation tree, and the data viewing panel located to the right of the Tivoli Performance Viewer navigation panel.

Refresh:

Click **Refresh** to rebuild the navigation tree. Refreshing is helpful when the available Performance Monitoring (PMI) Infrastructure data has changed, and the tree does not reflect those changes.

View Module(s):

Click **View Module** after one or more performance modules are selected in the tree to display the information for these modules in the data viewing panel.

The Data Monitoring panel enables the selection of multiple counters and displays the resulting performance data for the associated resources. It consists of two panels:

- Viewing Counter panel
- Counter Selection panel. If necessary, you can change the scaling factors by editing the default values in the scale field.

Deselect all items:

Select **Deselect all items** to quickly deselect all modules that are selected in the navigation tree.

Navigation tree:

Use the navigation tree to view advisor output, configure TPV, and select PMI modules for viewing.

- Click the **Advisor** node to have the advisor display the data in the viewing panel.
- Expand the **Settings** node to select and configure either the **User** or **Log** settings.
- Expand the **Performance Modules** node to view one or more PMI modules.

Advisor:

Click **Advisor** to examine various data while your application is running. The Performance Advisor provides advice to help tune systems for optimal performance using the PMI data collected.

The first table represents the number of requests per second and the response time in milliseconds for the Web container.

The pie graph displays the CPU activity as a percentage of busy and idle.

The third table displays average thread activity for the different resources, for example, Default, Object Request Broker, and Web container. Activity is expressed as the number of threads or connections busy and idle.

To view detailed information about the advice, select the message you want to view. This view provides additional information about the advice message, severity, description, user action, and detail.

User settings:

Change the values as needed for the following user settings:

Refresh Rate	Specifies how frequently TPV collects performance data for a server from the Performance Monitoring Infrastructure (PMI) service provided by that server. The default is 30 seconds. To collect performance data for the server more frequently, set the refresh rate to a smaller number. To collect performance data less frequently, set the refresh rate to a larger number. The allowed range is 5 to 500 seconds.
Buffer Size	Specifies the amount of data to be stored for a server. Data displayed in TPV is stored in a short in-memory buffer. After the buffer is full, each time a new entry is retrieved the oldest entry is discarded. The default buffer size is 40. Allowed values are 10, 20, 30, 40, 50, 60, 70, 80, 90 and 100. The larger the buffer size, the more memory is consumed. Thus, specify a buffer size that allows you capture enough monitoring data for analysis without wasting memory storing unneeded data.

View Data As	Specifies how counter values are displayed. Viewing options include the following: **Raw Value** Displays the absolute value. If the counter represents load data, for example, the average number of connections in a database pool, then TPV displays the current value. **Change in Value** Displays the change in the current value from the previous value. **Rate of Change** Displays the ratio `change/(T1 - T2)`, where *change* is the change in the current value from the previous value, *T1* is the time when the current value was retrieved, and *T2* is the time when the previous value was retrieved.

Log settings:

The log settings control what happens when **Start Logging** is clicked in, for example, a summary report on the performance of a servlet, enterprise bean (EJB), EJB method, connection pool or thread pool.

Change the value as needed for the following log settings:

Duration	Specifies the length of time, in minutes, that logging continues, unless **Stop Logging** is clicked first. TPV is not intended as a full-time logging solution.
Maximum File Size	Specifies the maximum size, in megabytes, of a single file. Note that TPV automatically zips log files to save space and this parameter controls the pre-zipped file size and not the post-zipped, which is smaller.
Maximum Number of Historical Files	Specifies the number of files TPV writes before stopping. If TPV reaches the maximum file size before the logging duration ends, it continues logging in another file, up to the maximum. If TPV reaches the maximum number of historical files before the logging duration ends, TPV deletes the oldest historical file and continues logging in a new file. The total amount of data that is stored is constrained by the **Maximum File Size** and **Maximum Number of Historical Files** parameters.
File Name	Specifies the name of the log file. The server name and the time at which the log is started is appended to the log name to help users identify a log file.
Log Output Format	Specifies whether TPV writes log files as XML or in a binary format. Binary format is recommended as it provides a smaller log file when not compressed.

View summary reports:

Summary reports are available for each application server.

Before viewing reports, make sure data counters are enabled and monitoring levels are set properly.

The standard monitoring level enables all reports except the report on Enterprise JavaBeans (EJB) methods. To enable an EJB methods report, adjust the PMI level to include EJB method data.

Tivoli Performance Viewer provides the following summary reports for each application server:

Servlets

The servlet summary lists all servlets that are running in the current application server. Use the servlet summary view to quickly find the most time intensive servlets and the applications that use them, and to determine which servlets are invoked most often. You can sort the summary table by any of the columns.

Tips

- Sort by **Avg Response Time** to find the slowest servlet or JavaServer page (JSP).
- Sort by **Total Requests** to find the servlet or JSP used the most.
- Sort by **Total Time** to find the servlet or JSP with the highest response times.

Enterprise JavaBeans

The Enterprise JavaBeans (EJB) summary lists all enterprise beans running in the server, the amount of time spent in their methods, the number of EJB invocations, and the total time spent in each enterprise bean.

```
total_time = number_of_invocations * time_in_methods
```

Sort the various columns to find the most expensive enterprise bean. Also, if the PMI counters are enabled for individual EJB methods, select the check box next to the EJB name see statistics for each of the methods.

Tips

- Sort by **Avg Response Time** to find the slowest enterprise beam.
- Sort by **Method Calls** to find the enterprise bean used the most.
- Sort by **Total Time** to find the enterprise bean with the slowest response time.

EJB Methods

The EJB method summary shows statistics for each EJB method. Use the EJB method summary to find the most costly methods of your enterprise beans.

Tips

- Sort by **Avg Response Time** to find the slowest EJB method.
- Sort by **Method Calls** to find the EJB method used the most.
- Sort by **Total Time** to find the EJB method with the slowest response time.

Connection pools

The connection pool summary lists all data source connections that are defined in the application server and shows their usage over time.

Tip

- When the application is experiencing normal to heavy usage, the pools used by that application should be nearly fully utilized. Low utilization means that resources are being wasted by maintaining connections or threads that are never used. Consider the order in which work progresses through the various pools. If the resources near the end of the pipeline are under utilized, it might mean that resources near the front are constrained or that more resources than necessary are allocated near the end of the pipeline.

Thread pools

The thread pool summary shows the usage of all thread pools in the application server over time.

Tip

- When the application is experiencing normal to heavy usage, the pools used by that application should be nearly fully utilized. Low utilization means that resources are being wasted by maintaining connections or threads that are never used. Consider the order in which work progresses through the various pools. If the resources near the end of the pipeline are under utilized, it might mean that resources near the front are constrained or that more resources than necessary are allocated near the end of the pipeline.

Performance module:

View performance modules that provide graphics and charts of various performance data on system resources such as CPU utilization, on WebSphere Application Server pools and queues such as database connection pools, and on customer application data such as servlet response time. In addition to providing a viewer for performance data, TPV enables you to view data for other products or customer applications that have implemented custom PMI.

Each performance 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 deselecting the check box next to them. By default, the first three counters for each module are shown.

Tivoli Performance Viewer displays interactive graphics using the Scalable Vector Graphics (SVG) format or non-interactive graphics using the JPG format. The SVG format is recommended because it provides a better user experience and is more processor and memory efficient for the application server.

In the performance module, you view current activity. This is a real time operation where the state of various system resources and their usage is displayed. Unless logging is turned on, data generated in this scenario will not be saved and is unavailable for subsequent viewing and analysis. To monitor behavior and system resources, click **Start Logging**. The user can replay and analyze the file at a later time.

Start Logging/Stop Logging

Use this to start or stop logging performance data. Once you start monitoring for your server, you will be able to view real time operation in the TPV panels.

Reset to Zero

This sets a new baseline using the current counter readings at the instant the button is clicked. Future data points are plotted on the graph relative to their position at the time **Reset to Zero** is clicked. Data points gathered prior to the time **Reset to Zero** is clicked are not displayed, although they are still held in the TPV buffer. If **Undo Reset to Zero** is clicked again, TPV displays all data currently in the buffer from their original baseline, not from the **Reset to Zero** point.

View Table/View Graph

To view the data in a table, click **View Table** on the counter selection table. To toggle back to a chart, click **View Graph**.

Show Legend/Hide Legend

To view the legend for a chart, click **Show Legend**. To hide the legend, click **Hide Legend**.

Clear Buffer

To clear values from the table or chart, click **Clear Buffer** beneath the chart or table. This removes all PMI data.

Unable to view Scalable Vector Graphics on Internet Explorer: If you encounter problems while viewing SVG with your Internet Explorer browser, visit the Adobe SVG website at http://www.adobe.com/svg/main.html to test your Adobe SVG Viewer. You can also download the most recent version of the Adobe SVG Viewer, view the release notes, and report any bugs with the Adobe SVG Viewer from this Web site.

Viewing TPV summary reports

The Tivoli Performance Viewer (TPV) provides five different summary reports that make important data quickly and easily accessible. View summary reports to help you find performance bottlenecks in your applications and modules. The TPV summary reports are generated upon request and are not dynamically refreshed.

Before you begin

This article assumes that one or more applications or modules are deployed and running on one or more servers.

About this task

In order to prepare a specific summary report, you must enable the minimum level of PMI data collection, or utilize the **custom** level of monitoring and enable specific counters.

Use the administrative console to view TPV summary reports.

1. Click **Monitoring and Tuning > Performance Viewer > Current Activity >** *server_name* **> Summary Reports** in the console navigation tree.
2. Select the code artifact or pool for which you want a summary report. Expand the **Summary Reports** node of the TPV navigation tree on the left side of the Tivoli Performance Viewer page to see links for the types of summary reports. After clicking on a link for an artifact or pool, a list of artifacts or pools on the server is displayed on the right side of the page.
3. Select the artifact or pool for which you want to view a summary report.

Tivoli Performance Viewer summary report types:

Servlets

The servlet summary lists all servlets that are running in the current application server. Use the servlet summary view to quickly find the most time intensive servlets and the applications that use them, and to determine which servlets are invoked most often. You can sort the summary table by any of the columns.

Tips

- Sort by **Avg Response Time** to find the slowest servlet or JavaServer page (JSP).
- Sort by **Total Requests** to find the servlet or JSP used the most.
- Sort by **Total Time** to find the most costly servlet or JSP.

Enterprise beans

The Enterprise JavaBeans (EJB) summary lists all enterprise beans running in the server, the amount of time spent in their methods, the number of EJB invocations, and the total time spent in each enterprise bean.

```
total_time = number_of_invocations * time_in_methods
```

Sort the various columns to find the most expensive enterprise bean. Also, if the PMI counters are enabled for individual EJB methods, there is a check box next to the EJB name that you can select to see statistics for each of the methods.

Tips

- Sort by **Avg Response Time** to find the slowest enterprise beam.
- Sort by **Method Calls** to find the enterprise bean used the most.
- Sort by **Total Time** to find the most costly enterprise bean.

EJB methods

The EJB method summary shows statistics for each EJB method. Use the EJB method summary to find the most costly methods of your enterprise beans.

Tips
- Sort by **Avg Response Time** to find the slowest EJB method.
- Sort by **Method Calls** to find the EJB method used the most.
- Sort by **Total Time** to find the most costly EJB method.

Connection pools

The connection pool summary lists all data source connections that are defined in the application server and shows their usage over time.

Tip
- When the application is experiencing normal to heavy usage, the pools used by that application should be nearly fully utilized. Low utilization means that resources are being wasted by maintaining connections or threads that are never used. Consider the order in which work progresses through the various pools. If the resources near the end of the pipeline are under utilized, it might mean that resources near the front are constrained or that more resources than necessary are allocated near the end of the pipeline.

Thread Pools

The thread pool summary shows the usage of all thread pools in the application server over time.

Tip
- When the application is experiencing normal to heavy usage, the pools used by that application should be nearly fully utilized. Low utilization means that resources are being wasted by maintaining connections or threads that are never used. Consider the order in which work progresses through the various pools. If the resources near the end of the pipeline are under utilized, it might mean that resources near the front are constrained or that more resources than necessary are allocated near the end of the pipeline.

PMI levels and counters required

In order to view Tivoli Performance Viewer (TPV) summary reports, the minimum PMI level must be enabled. Otherwise, you must use the **custom** monitoring level, and enable the PMI level counters required for the specific report you want to view.

Table 1. Required properties for TPV summary reports

Summary Report	PMI level required	Custom PMI counters required
Servlets	Basic	JDBC Connection Pools.PoolSize JDBC Connection Pools.AllocateCount JDBC Connection Pools.ReturnCount
Enterprise beans	Basic	Thread Pools.PoolSize Thread Pools.ActiveCount
EJB methods	All	Enterprise Beans.MethodCallCount Enterprise Beans.MethodResponseTime
Connection pools	Extended	WSEJBStats.MethodStats.MethodLevelCallCount WSEJBStats.MethodStats.MethodLevelResponseTime

Table 1. Required properties for TPV summary reports (continued)

Thread pools	Extended	Web Applications.RequestCount Web Applications.ServiceTime

Related tasks

"Enabling PMI data collection" on page 174
Enable PMI data collection to diagnose problems and tune application performance.

"Viewing TPV summary reports" on page 233
The Tivoli Performance Viewer (TPV) provides five different summary reports that make important data quickly and easily accessible. View summary reports to help you find performance bottlenecks in your applications and modules. The TPV summary reports are generated upon request and are not dynamically refreshed.

Viewing PMI data with TPV

You can use the Tivoli Performance Viewer (TPV) to view Performance Monitoring Infrastructure (PMI) data in chart or table form.

Before you begin

TPV monitors the performance activity of all servers on a node. This article assumes that one or more servers have been created and are running on the node, and that PMI is enabled.

TPV displays graphics in either the Scalable Vector Graphics (SVG) format or as a static image in the JPG format. If you do not have the Adobe SVG browser plug-in installed, you are prompted to download and install it. If you select not to install the plug-in (by selecting **Cancel**), TPV displays the static image. Installing the Adobe SVG plug-in is advantageous for several reasons. First, the SVG format provides interactive graphics that provide additional information when you hover your mouse over a point, line, or legend item. The SVG format also enables you to click a point and see details for it. Secondly, using the SVG format provides a performance benefit because the work to display the SVG image is done on the client side. When viewing a static image, the application server must convert the SVG image into a static image, which is a processor and memory intensive operation. If your browser is Internet Explorer 7, the Adobe SVG installation prompt might be inaccessible. To resolve the problem, reinstall Adobe SVG.

If you experience problems with SVG, see "Scalable Vector Graphics problems" on page 236 for more information.

About this task

You view performance modules when your server is experiencing performance problems. For example, a common performance problem occurs when individual sessions are too large. To help view data on a session, you can view the Servlet Session Manager PMI module and monitor the **SessionObjectSize** counter to make sure that **Session Object Size** is not too large.

Performance modules are shown in the TPV current activity settings in the administrative console.

Note: When you create a second application server under WebSphere Application Server, TPV cannot display the PMI data for the second application server such as WebSphere Portal Server. For more information, refer to the technote at: http://www.ibm.com/support/docview.wss?rs=180 &=swg21247170.

- Select PMI data to view.
  1. Click **Monitoring and Tuning > Performance Viewer > Current Activity >** *server_name* **> Performance Modules** in the console navigation tree.
  2. Place a check mark in the check box beside the name of each performance module that you want to view. Expand the tree by clicking **+** next to a node and shrink it by clicking **–** next to a node.

If you do not see all the PMI counters you expect, or a PM you are interested in is visible but cannot be selected, PMI is not enabled for that particular counter or for any counter in that PM. Go to the PMI control area of the administrative console and enable PMI for any counters you wish to view, then return to TPV and select those PMs. To view the PMI page, click on **Monitoring and Tuning > Performance Monitoring Infrastructure >** *server_name*.

3. Click on **View Modules**. A chart or table providing the requested data is displayed on the right side of the page. Charts are displayed by default.

   Each module has several counters associated with it. These counters are displayed in a table underneath the data chart or table. Selected counters are displayed in the chart or table. You can add or remove counters from the chart or table by selecting or deselecting individual counters. 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. The data from all the PMI counters that are currently enabled in each PM for all active instances of the PM can be recorded and captured in a TPV log file. Refer to "Logging performance data with TPV" on page 237 for more information on the 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 data in a server during a particular period of time.
4. Optional: To remove a module from a chart or table, deselect the check box next to the module and click **View Modules** again.
5. 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**.
6. Optional: To view the legend for a chart, click **Show Legend**. To hide the legend, click **Hide Legend**.

- Scale the PMI data. You can manually adjust the scale for each counter so that the graph displays meaningful comparisons of different counters.
  1. Find the counter whose scale you want to modify in the table beneath the chart.
  2. Change the value for **Scale** as needed. **Tips:**
     – When the scale is set to 1 the true value of the counter is displayed in the graph.
     – A value greater than 1 indicates that the value is amplified by the factor shown.
     – A value less than 1 indicates that the variable is decreased by the factor shown.

     For example, a scale setting of .5 means that the counter is graphed as one-half its actual value. A scale setting of 2 means that the counter is graphed as twice its actual value. Scaling only applies to the graphed values and has no effect on the data displayed when viewing it in table form.
  3. Click **Update**.
- Clear values from tables and charts.
  1. Ensure that one or more modules are selected under **Performance Modules** in the TPV navigation tree
  2. Click **Clear Buffer** beneath the chart or table. The PMI data is removed from a table or chart.
- Reset counters to zero (0).
  1. Ensure that one or more modules are selected under **Performance Modules** in the TPV navigation tree
  2. Click **Reset to Zero** beneath the chart or table. **Reset to Zero** sets a new baseline using the current counter readings at the instant the button is clicked. Future datapoints are plotted on the graph relative to their position at the time **Reset to Zero** is clicked. Datapoints gathered prior to the time **Reset to Zero** is clicked are not displayed, although they are still held in the TPV buffer. If **Undo Reset to Zero** is clicked again, TPV displays all data recorded from the original baseline, not from the **Reset to Zero** point.

     For information on how TPV displays the data, see "Configuring TPV settings" on page 226.

Scalable Vector Graphics problems:

Consult this topic for problems and solutions for Scalable Vector Graphics.

Using TPV without Scalable Vector Graphics (SVG)

When viewing performance data in Tivoli Performance Viewer (TPV) on a browser that does not support Scalable Vector Graphics (SVG) graphics, either using the Adobe SVG plug-in or through native support, complex graphs might take a long time to refresh. Eventually an out of memory error may occur.

When TPV detects a browser that does not support SVG, it uses Apache Batik to transcode the SVG into a binary image format. The transcoding process is highly memory intensive. The complexity of the graph is affected by two things: the TPV buffer size, which dictates how many points are in each line that is drawn, and the number of selected data points, which dictates how many lines are drawn.

To avoid long refresh times or the out of memory errors, use a browser that supports SVG, either with a plug-in, like the Adobe SVG plug-in, or natively, which is included in some Mozilla builds. If this is not possible, keep the graph simple by limiting the size of the buffer and the number of selected data points.

Unable to view Scalable Vector Graphics on Internet Explorer

If you encounter problems while viewing SVG with your Internet Explorer browser, visit the Adobe SVG Zone Web site at http://www.adobe.com/svg/main.html to test your Adobe SVG Viewer. You can also download the most recent version of the Adobe SVG Viewer, view the release notes, and report any bugs with the Adobe SVG Viewer from this Web site.

Logging performance data with TPV

The Tivoli Performance Viewer (TPV) provides an easy way to store real-time data for system resources, WebSphere Application Server pools and queues, and applications in log files for later retrieval. You can start and stop logging while viewing current activity for a server, and later replay this data. Logging of performance data captures performance data in windows of time so you can later analyze the data.

Before you begin

This article assumes that one or more servers have been created and are running on the node, and that you have configured the TPV log settings. The log settings may affect performance and are described in detail in “Viewing Data with the Tivoli Performance Viewer” on page 228. The TPV logging feature is not intended to be a full-time monitoring solution, but instead for selective data recording for subsequent replay and analysis.

About this task

You can study the sequence of events that led to a peculiar condition in the application server.

First, enable TPV logging so performance data generated in the application server persists in a log file stored at a specific location. Later, using the replay feature in TPV, view the performance data that was generated in exactly the same chronological order as it was generated in real time, enabling you to analyze a prior sequence of events.

You do not need to know the syntax and format in which log files are generated and stored. Do not edit log files generated by TPV; doing so will irrecoverably corrupt or destroy the performance data stored in the log files.

You can create and view logs in the administrative console.

Note: Performance degradation can occur when you use the Mozilla Firefox browser and Tivoli Performance Viewer to monitor the Application Server. Additionally, a memory leak problem occurs when you use Tivoli Performance Viewer for an extended period of time with the Mozilla Firefox browser.

Use Microsoft® Internet Explorer with Tivoli Performance Viewer to monitor the Application Server. To use Internet Explorer, you must have a Scalable Vector Graphics (SVG) Viewer installed as a plug-in. The Mozilla Firefox browser provides a build-in plug-in. It is recommended that you use Internet Explorer when you are using the Tivoli Performance Viewer for long-term or extended monitoring.

- Create logs.
  1. Click **Monitoring and Tuning > Performance Viewer > Current Activity >** *server_name* **> Settings > Log** in the console navigation tree. To see the **Log** link on the Tivoli Performance Viewer page, expand the **Settings** node of the TPV navigation tree on the left side of the page. After clicking **Log**, the TPV log settings are displayed on the right side of the page.
  2. Click on **Start Logging** when viewing summary reports or performance modules.
  3. When finished, click **Stop Logging** . Once started, logging stops when the logging duration expires, **Stop Logging** is clicked, or the file size and number limits are reached. To adjust the settings, see step 1.

  By default, the log files are stored in the *profile_root*/`logs/tpv` directory on the node on which the server is running. TPV automatically compresses the log file when it finishes writing to it to conserve space. At this point, there must only be a single log file in each `.zip` file and it must have the same name as the `.zip` file.
- View logs.
  1. Click **Monitoring and Tuning > Performance Viewer > View Logs** in the console navigation tree.
  2. Select a log file to view using either of the following options:

     Explicit Path to Log File

     Choose a log file from the machine on which the browser is currently running. Use this option if you have created a log file and transferred it to your system. Click **Browse** to open a file browser on the local machine and select the log file to upload.

     Server File

     Specify the path of a log file on the server.

     In a stand-alone application server environment, type in the path to the log file. The *profile_root*`\logs\tpv` directory is the default on a Windows system.

     In a deployment manager environment, click the **Browse** button next to the input to browse the various nodes and find the log file to view.
  3. Click **View Log**. The log is displayed with log control buttons at the top of the view.
  4. Adjust the log view as needed. Buttons available for log view adjustment are described below. By default, the data replays at the **Refresh Rate** specified in the user settings. You can choose one of the **Fast Forward** modes to play data at rate faster than the refresh rate.

Rewind	Returns to the beginning of the log file.
Stop	Stops the log at its current location.
Play	Begins playing the log from its current location.
Fast Forward	Loads the next data point every three (3) seconds.
Fast Forward 2	Loads ten data points every three (3) seconds.

You can view multiple logs at a time. After a log has been loaded, return to the View Logs panel to see a list of available logs. At this point, you can load another log.

TPV automatically compresses the log file when finishes writing it. The log does not need to be decompressed before viewing it, though TPV can view logs that have been decompressed.

Viewing Data Recorded by the TPV

Use this page to view logged data from Tivoli Performance Viewer.

To view this administrative console page, click **Monitoring and Tuning** > **Performance Viewer** > **View Logs**.

Explicit path to log file:

Select explicit path to a log file, specify the path name, and click **View Log** to display the stored data.

Server file:

Select server file, specify the path name, and click **View Log** to display the stored data.

Capturing a RMF workload activity report

Using the Workload Activity Reports, you can observe the measurements in the service and reporting classes.

Before you begin

For more information on RMF workload activity reports and the RMF Monitor III refer to the article “RMF Workload Activity reports and RMF Monitor III.”

About this task

The following RMF1 job reads the data from RMF data buffers.

1. Run the RMF Monitor 1 post processor as a batch job which can read the RMF data buffers from memory and produce a report.

   The sample job that follows will run the RMF Monitor 1 post processor (You will need to change the time and date parameters.)

   ```
   //RMF1JOB  JOB 1,CLASS=A
   //RMFPP    EXEC PGM=ERBRMFPP,REGION=0M
   //MFPMSGDS DD   SYSOUT=*
   //SYSIN    DD   *
    SYSOUT(0)
    NOSUMMARY
    SYSRPTS(WLMGL(SCPER(WSHIGH,WSMED,WSLOW,SYSSTC,OPS_DEF)))
    DATE(04172003,04172003)        /* <== SET TO MEAS. DATE  MMDDYYYY  */
    RTOD(1430,1500)                /* <== SET TO MEAS. TIME OF DAY HHMM*/
    DINTV(0002)
   /*
   //
   ```

   This will take the raw SMF data, in this case the MANE data, and produce the workload activity report as an H output class in SDSF.

2. Using the Workload Activity Reports, observe the measurements in the service and reporting classes. After your measurement run is over, you should reset the SMF parameters to their standard settings with the `SET SMF=xx` command.

RMF Workload Activity reports and RMF Monitor III

Performance metrics include transaction rates and response times. Resource utilization includes CPU, I/O (channel), and storage utilization.

Transactions/second

This is shown in the AVG, MPL, and AVG ENC fields which is equal to the average number of enclaves in the period in the RMF Monitor I report below.

Response times

The actual response times of the WLM transaction is shown in the TRANS.-TIME SS.TTT' column n the RMF Monitor I report below and are measured in milliseconds. (This also includes time waiting on WLM queue.)

Service Rates

Resource utilization in CPU service units, and Service Units per Sec. The APPL% field shows the number of processor engines (CPs) required to drive the work in the service (report) class.

Controller Regions Example: Here is the report for the WebSphere controller regions which were all assigned a particular reporting class according to the STC classification rules based on the started class jobname. There is only one transaction active in the system, there are no response time figures, and required 37.2% of a processor engine to support its work.

```
TRANSACTIONS    TRANS.-TIME  SS.TTT   ---SERVICE--   --SERVICE RATES--
AVG      1.00   ACTUAL            0   IOC        0   ABSRPTN     89615
MPL      1.00   EXECUTION         0   CPU   522567   TRX SERV    89615
ENDED       0   QUEUED            0   MSO    10159K  TCB          39.9
END/S    0.00   R/S AFFINITY      0   SRB    61728   SRB           4.7
#SWAPS      0   INELIGIBLE        0   TOT    10743K  RCT           0.0
EXCTD       0   CONVERSION        0   /SEC   89630   IIT           0.0
AVG ENC  0.00   STD DEV           0                  HST           0.0
REM ENC  0.00                                      APPL %       37.2
```

Servant Regions Example: Here is the report for the WebSphere servant regions which were all assigned a particular reporting class according to the STC classification rules based on the stared class jobname. There are only two transactions active in the system (meaning there were two servant regions), there are no response time figures, and they required 10% of a processor engine to support their work.

```
TRANSACTIONS    TRANS.-TIME  SS.TTT   ---SERVICE--   --SERVICE RATES--
AVG      2.00   ACTUAL            0   IOC        0   ABSRPTN    122075
MPL      2.00   EXECUTION         0   CPU   143957   TRX SERV   122075
ENDED       0   QUEUED            0   MSO    29113K  TCB          11.0
END/S    0.00   R/S AFFINITY      0   SRB    12460   SRB           1.0
#SWAPS      0   INELIGIBLE        0   TOT    29270K  RCT           0.0
EXCTD       0   CONVERSION        0   /SEC  244192   IIT           0.0
AVG ENC  0.00   STD DEV           0                  HST           0.0
REM ENC  0.00                                      APPL %       10.0
```

WebSphere Transactions (Enclaves) Example: Here is the report for the real WebSphere transaction work which runs as enclaves in this particular reporting class. The average number of these kind of transactions active in the system was 241.52, the average response time was 276 milliseconds, and required 2.129 processor engines to support this work.

```
TRANSACTIONS    TRANS.-TIME  SS.TTT   ---SERVICE--   --SERVICE RATES--
AVG    241.52   ACTUAL          276   IOC        0   ABSRPTN       115
MPL    241.52   EXECUTION       272   CPU     3343K  TRX SERV      115
ENDED  106717   QUEUED            4   MSO        0   TCB         255.5
END/S  890.32   R/S AFFINITY      0   SRB        0   SRB           0.0
#SWAPS      0   INELIGIBLE        0   TOT     3343K  RCT           0.0
EXCTD       0   CONVERSION        0   /SEC      17   IIT           0.0
AV ENC 241.52   STD DEV          66                  HST           0.0
REM ENC  0.00                                      APPL %      212.9
```

Delays Example: The QMPL field in the following report means that the server was waiting for a Server Region to select work off the WLM queue. Here is a queue delay report which shows that work was delayed 23.5% of the time waiting for the CPU and 12.6% of the time waiting for a servant region:

```
           EX   PERF   AVG   --USING%--  ----- EXECUTION DELAYS % -----
           VEL  INDX  ADRSP   CPU   I/O  TOTAL   CPU QMPL
GOAL     40.0%
ACTUALS 45.3%  .89   13.4   0.1   0.0   36.1  23.5 12.6
```

Started Class Classification Example: Here is an example of the STC classification rules/panel in WLM which can be used to classify the non-enclave time in the websphere server regions. WebSphere production controller region with a jobname of WSPRODC is assigned a service class of STCCR and reporting class of RCTLREG, production controller region with a jobname of WSPRODC is assigned a service class of STCSR and reporting class of RSRVREG, and testregions (servant and controllers) with a jobname starting with WSTare assigned a service class of STCWSTST and reporting class of RWSTST.

```
 Subsystem Type . . . . . : STC
 Description  . . . . . . . . . . All started tasks
   -------Qualifier-------------            -------Class--------
   Type       Name     Start                Service     Report
   Type       Name     Start                Service     Report
                                  DEFAULTS: STCMED__    RSTCMED_
1  TNG        WSPRODC  ___                  STCCR___    RCTLREG_
1  TNG        WSPRODS  ___                  STCSR___    RSRVREG_
1  TNG        WST*     ___                  STCWSTST    RWSTST__
```

Third-party performance monitoring and management solutions

Several performance monitoring, problem determination, and management solutions are available that can be used with WebSphere Application Server.

These products use WebSphere Application Server interfaces, including:

- Performance Monitoring Infrastructure (PMI)
- Java Management Extensions (JMX)

See the topic “Performance: Resources for learning” on page 1 for a link to IBM business partners providing monitoring solutions for WebSphere Application Server.

WLM Delay Monitoring

WebSphere Application Server for z/OS uses Workload Manager (WLM) services to report transaction begin-to-end response times and execution delay times.

The WLM data collected by Resource Measurement Facility (RMF) is captured in two phases of the RMF report:

- BTE - the begin-to-end phase applies to requests handled by the controller
- EXE - the execution phase applies to requests handled by the servant

You can use this status information to determine where possible performance bottlenecks are occurring. This feature is available on z/OS V1R2 and above with WLM APAR OW51848 and RMF APAR OW52227.

For details on workload management, see *z/OS MVS™ Planning: Workload Management*. For details on WLM delay monitoring, refer to *z/OS MVS Programming: Workload Management Services*. Both are available on the z/OS Internet Library Web site.

When a new transaction enters the system, the WebSphere Application Server for z/OS application control region (ACR) starts the classify service. Delays associated with the WebSphere Application Server for z/OS ACR service class are counted separately for the BTE phase and the EXE phase. This support allows WLM to associate a performance block (PB) with an enclave to record delays that occur in the flow of a transaction. The state samples are collected on an ongoing basis and reported as a percentage of

average transaction response time. The following table shows the states, their codes, the section of the RMF report where each is reported, the meaning, and suggested response. You can use this information in the RMF report to determine where some of your system's performance problems may be occurring.

Table 2. WLM delay monitoring states

State	Code	Report	Meaning	Response
ACTIVE	ACTIVE SUB	Both BTE and EXE	WebSphere is actively processing request	
ACTIVE_APPLIC	ACTIVE APPL	Both BTE and EXE	Application is running	Use application monitoring tool to determine the cause of the delay.
WAITING TYPE1	TYP1	EXE	EJB collaborator delay	
WAITING TYPE2	TYP2	EXE	Resource manager delay	Call a J2C connector to resource managers such as DB2, IMS, and CICS. Investigate the other resource manager using their monitoring tools.
WAITING TYPE3	TYP3	EXE	Servant called to a different distributed object server using RMI/IIOP	1. Investigate the delay on the other server. The delay might point to session caches. 2. Look for any network problems. 3. Avoid outbound calls.
WAITING TYPE 4	TYP4	BTE	OTS call to RRS. Occurs only in controller when controller is trying to commit a distributed transaction.	1. Investigate the delay on the other server. 2. Look for any network problems. 3. Consider combining application into one server to avoid delay.
WAITING REGIST TO WORKTABLE	WORK	BTE	An indication of contention within the controller while trying to process concurrent requests.	If delay is excessive, consider adding another controller and splitting work off to it.
WAITING OTHER_PRODUCT	OTHER	BTE	Indicates a configuration problem in DNS or TCP/IP	Check to make sure all the DNS servers are running. You might want to look at OPING or ONSLOOKUP.
WAITING DISTRIB	DIST	BTE	Controller as a client went outbound waiting for a response.	1. Investigate the delay on the other server. 2. Look for any network problems. 3. Consider combining application into one server to avoid delay.
WAITING SESS_NETWORK	REMT	BTE	Time spent waiting for a TCP/IP session to be established on the network.	The two session delays should be observable in conjunction with TYP3 delays. Look at TCP/IP configuration.

Table 2. WLM delay monitoring states (continued)

WAITING SESS_SYSPLEX	SYSP	BTE	Time spent waiting for a TCP/IP session to be established on the sysplex.	The two session delays should be observable in conjunction with TYP3 delays. Look at TCP/IP configuration.
WAITING REGULAR_THREAD	REGT	BTE	Waiting for a thread in the controller. Work is bottlenecked in the controller because it is receiving more requests than it can process.	Split the controller.
WAITING SSL_THREAD	SSLT	BTE	Waiting for an SSL thread in the controller. Work is bottlenecked in the controller because it is receiving more requests for SSL handshakes than it can process.	Split controller to increase SSL threads. 1. Increase SSL threads. 2. Look at SSL configuration. 3. Split the controller to increase SSL threads.
WAITING SESS_LOCALMVS	LOCL	BTE	Time spent communicating with a different distributed object server using local optimized communication.	1. Investigate the delay on the other server. 2. Avoid outbound calls.

RMF report examples

The Workload Manager data collected by Resource Measurement Facility (RMF) is captured and displayed in a report.

This section includes several examples of RMF reports with WLM delay monitoring information.

```
            RESP  ---------------------- STATE SAMPLES BREAKDOWN (%) ---------  ------STATE------
SUB    P    TIME  --ACTIVE-- READY IDLE  ------WAITING FOR-----------------  SWITCHED SAMPL(%)
TYPE         (%)   SUB  APPL              TYP1                               LOCAL SYSPL REMOT
CB     BTE   0.0   0.0   0.0   0.0  0.0   0.0                                 0.0   0.0   0.0
CB     EXE  95.7   0.5  74.7   0.0  0.0  24.9                                 0.0   0.0   0.0

            RESP  ---------------------- STATE SAMPLES BREAKDOWN (%) ---------  ------STATE------
SUB    P    TIME  --ACTIVE-- READY IDLE  ------WAITING FOR-----------------  SWITCHED SAMPL(%)
TYPE         (%)   SUB  APPL              TYP4 REGT LOCL                     LOCAL SYSPL REMOT

CB     BTE   0.0  26.9   0.0   0.0  0.0  65.4  3.8  3.8                      0.0   0.0   0.0
CB     EXE   0.0   0.0   0.0   0.0  0.0   0.0  0.0  0.0                      0.0   0.0   0.0

            RESP  ---------------------- STATE SAMPLES BREAKDOWN (%) ---------  ------STATE------
SUB    P    TIME  --ACTIVE-- READY IDLE  ------WAITING FOR-----------------  SWITCHED SAMPL(%)
TYPE         (%)   SUB  APPL                                                 LOCAL SYSPL REMOT
CB     BTE   0.0   100   0.0   0.0  0.0                                       0.0   0.0   0.0
CB     EXE   0.0   0.0   0.0   0.0  0.0                                       0.0   0.0   0.0

            RESP  ---------------------- STATE SAMPLES BREAKDOWN (%) ---------  ------STATE------
SUB    P    TIME  --ACTIVE-- READY IDLE  ------WAITING FOR-----------------  SWITCHED SAMPL(%)
TYPE         (%)   SUB  APPL              TYP4 REGT SYSP                     LOCAL SYSPL REMOT
CB     BTE   0.0  40.7   0.0   0.0  0.0  51.9  3.7  3.7                      0.0   0.0   0.0
CB     EXE   0.0   0.0   0.0   0.0  0.0   0.0  0.0  0.0                      0.0   0.0   0.0

            RESP  ---------------------- STATE SAMPLES BREAKDOWN (%) ---------  ------STATE------
SUB    P    TIME  --ACTIVE-- READY IDLE  ------WAITING FOR-----------------  SWITCHED SAMPL(%)
```

```
TYPE          (%)   SUB  APPL              TYP4 REGT WORK                    LOCAL SYSPL REMOT
CB    BTE    0.0  40.0   0.0   0.0  0.0  50.0  7.5  2.5                     0.0   0.0   0.0
CB    EXE    0.0   0.0   0.0   0.0  0.0   0.0  0.0  0.0                     0.0   0.0   0.0
```

Chapter 4. Monitoring application flow

Monitoring, optimizing, and troubleshooting WebSphere Application Server performance can be a challenge. This article gives you a basic strategy for monitoring with an understanding of the application view.

About this task

This information includes understanding the application flow that satisfies the end user request. This perspective provides the views of specific servlets that access specific session beans, entity container-managed persistence beans, and a specific database. This perspective is important for the in-depth internal understanding of who is using specific resources. Typically at this stage, you deploy some type of trace through the application, or thread analysis under load condition techniques to isolate areas of the application and particular interactions with the back-end systems that are especially slow under load. In this case, WebSphere Application Server provides request metrics to help trace each individual transaction as it flows through the application server, recording the response time at different stages of the transaction flow (for example, request metrics records the response times for the Web server, the Web container, the Enterprise JavaBeans container, and the back-end database). In addition, several IBM development and monitoring tools that are based on the request metrics technology (for example, Tivoli Monitoring for Transaction Performance) are available to help view the transaction flow.

Why use request metrics?

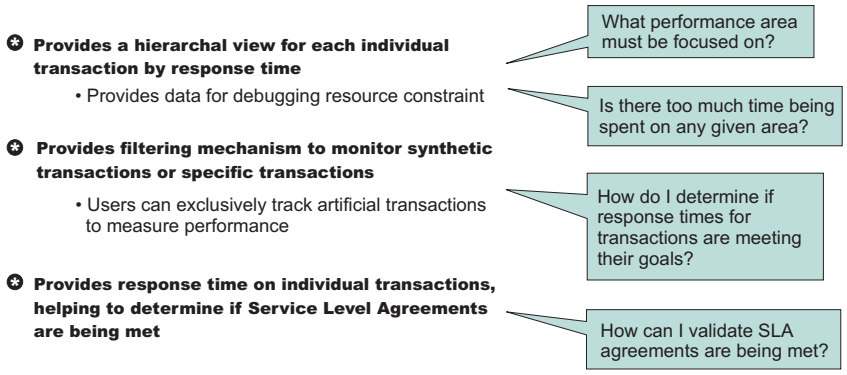
Request metrics is a tool that enables you to track individual transactions, recording the processing time in each of the major WebSphere Application Server components.

Before you begin

The information that is tracked by request metrics might either be saved to log files for later retrieval and analysis, be sent to Application Response Measurement (ARM) agents, or both.

As a transaction flows through the system, request metrics includes additional information so that the log records from each component can be correlated, building up a complete picture of that transaction. The result looks similar to the following example:

```
HTTP request/trade/scenario ------------------------------> 172 ms
     Servlet/trade/scenario  ------------------------------> 130 ms
         EJB TradeEJB.getAccountData ---------------------->  38 ms
              JDBC select ------------------------------->   7 ms
```



This transaction flow with associated response times can help you target performance problem areas and debug resource constraint problems. For example, the flow can help determine if a transaction spends

most of its time in the Web server plug-in, the Web container, the Enterprise JavaBeans (EJB) container or the back end database. The response time that is collected for each level includes the time spent at that level and the time spent in the lower levels. For example, the response time for the servlet, which is 130 milliseconds, also includes 38 milliseconds from the enterprise beans and Java Database Connectivity. Therefore, 92 ms can be attributed to the servlet process.

Request metrics tracks the response time for a particular transaction. Because request metrics tracks individual transactions, using it imposes some performance implications on the system. However, this function can be mitigated by the use of the request filtering capabilities.

For example, tools can inject synthetic transactions. Request metrics can then track the response time within the WebSphere Application Server environment for those transactions. A synthetic transaction is one that is injected into the system by administrators to take a proactive approach to testing the performance of the system. This information helps administrators tune the performance of the Web site and take corrective actions. Therefore, the information that is provided by request metrics might be used as an alert mechanism to detect when the performance of a particular request type goes beyond acceptable thresholds. The filtering mechanism within request metrics might be used to focus on the specific synthetic transactions and can help optimize performance in this scenario.

About this task

When you have the isolated problem areas, use request metrics filtering mechanism to focus specifically on those areas. For example, when you have an isolated problem in a particular servlet or EJB method, use the uniform resource identifier (URI) algorithms or EJB filter to enable the instrumentation only for the servlet or EJB method. This filtering mechanism supports a more focused performance analysis.

Five types of filters are supported:
- Source IP filter
- URI filter
- EJB method name filter
- JMS parameters filter
- Web services parameters filter

When filtering is enabled, only requests that match the filter generate request metrics data, create log records, call the ARM interfaces, or all. You can inject the work into a running system (specifically to generate trace information) to evaluate the performance of specific types of requests in the context of a normal load, ignoring requests from other sources that might be hitting the system.

Note: Filters are only applicable where the request first enters WebSphere Application Server.
- Learn more about request metrics by reviewing detailed explanations in this section.
- Configure and enable request metrics.
- Retrieve performance data and monitor application flow.
- Extend monitoring to track applications that might require additional instrumentation points.

Example: Using request metrics

Use this page to learn how to use request metrics by viewing an example.

In this example, the HitCount servlet and the Increment enterprise bean are deployed on two different application server processes. As shown in the following diagram, the Web container tier and Enterprise JavaBeans (EJB) container tiers are running in two different application servers. To set up such a configuration, install WebSphere Application Server Network Deployment.

Assume that the Web server and the Web container tier both run on machine 192.168.0.1, and the Enterprise JavaBeans (EJB) container tier runs on a second machine 192.168.0.2. The client requests might be sent from a different machine; 192.168.0.3, for example, or other machines.

To illustrate the use of source IP filtering, one source IP filter (192.168.0.3) is defined and enabled. You can trace requests that originate from machine 192.168.0.3 through http://192.168.0.1/hitcount?selection=EJB&lookup=GBL&trans=CMT. However, requests that originate from any other machines are not traced because the source IP address is not in the filter list.

By only creating a source IP filter, any requests from that source IP address are effectively traced. This tool is effective for locating performance problems with systems under load. If the normal load originates from other IP addresses, then its requests are not traced. By using the defined source IP address to generate requests, you can see performance bottlenecks at the various hops by comparing the trace records of the loaded system to trace records from a non-loaded run. This ability helps focus tuning efforts to the correct node and process within a complex deployment environment.

Make sure that request metrics is enabled using the administrative console. Also, make sure that the trace level is set to at least hops (writing request traces at process boundaries). Using the configuration previously listed, send a request http://192.168.0.1/hitcount?selection=EJB&lookup=GBL&trans=CMT through the HitCount servlet from machine 192.168.0.3.

In this example, at least three trace records are generated:

- A trace record for the Web server plug-in is displayed in the plug-in log file (default location is (*plugins_root*/logs/*web_server_name*/http_plugin.log) on machine 192.168.0.1.
- A trace record for the servlet displays in the application server log file (default location is *profile_root*/logs/appserver/SystemOut.log) on machine 192.168.0.1.
- A trace record for the increment bean method invocation displays in the application server log file (default location is *profile_root*/logs/appserver/SystemOut.log) on machine 192.168.0.2.

The two trace records that are displayed on machine 192.168.0.1 are similar to the following example:

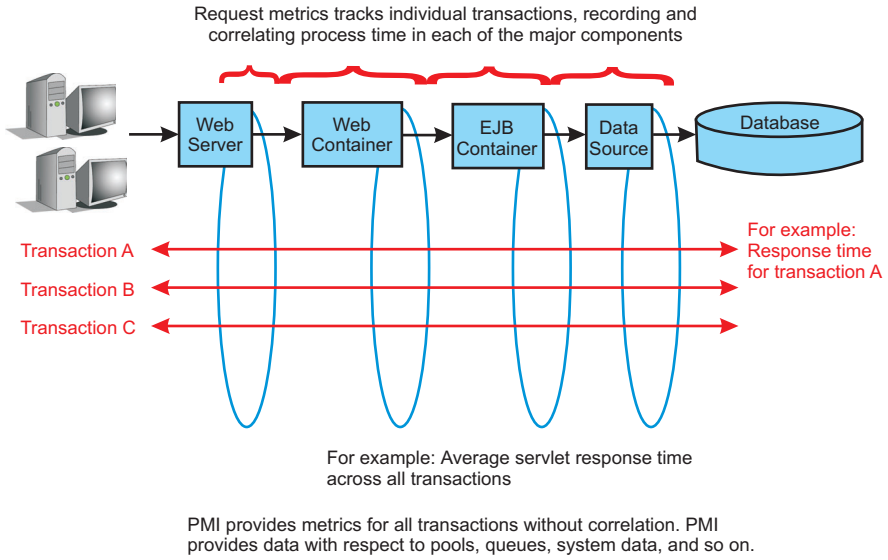
```
PLUGIN:
parent:ver=1,ip=192.168.0.1,time=1016556185102,pid=796,reqid=40,event=0
- current:ver=1,ip=192.168.0.1,time=1016556185102,pid=796,reqid=40,event=1
type=HTTP detail=/hitcount elapsed=90 bytesIn=0 bytesOut=2252

Application server (web container tier)
PMRM0003I: parent:ver=1,ip=192.168.0.1,time=1016556185102,pid=796,reqid=40,event=0
- current:ver=1,ip=192.168.0.1,time=1016556186102,pid=884,reqid=40,event=1
type=URI detail=/hitcount elapsed=60
```

The trace record that is displayed on machine 192.168.0.2 is similar to the following example:

```
PMRM0003I:
parent:ver=1,ip=192.168.0.1,time=1016556186102,pid=884,reqid=40,event=1
-
current:ver=1,ip=192.168.0.2,time=1016556190505,pid=9321,reqid=40,event=1
type=EJB
detail=com.ibm.defaultapplication.Increment.increment elapsed=40
```

Data you can collect with request metrics

Typically, different components of the enterprise application might be hosted across several nodes in a distributed system. For example, the servlets might be hosted on one node, while the enterprise beans on which these servlets depend might be hosted on an entirely different node. When a request comes to a process, the process might send the request to one or more downstream processes, as shown in the following figure:



Trace records might be generated for each process with associated elapsed times for that process. These trace records might be correlated together to build a complete picture of the request flow through the distributed system, similar to the diagram in “Why use request metrics?” on page 245.

You can view the process response time that is monitored by request metrics through the Application Response Measurement (ARM) interface and system log files. When a request is sent to the application server, request metrics captures response times for the initiating request and any related downstream invocations. Request metrics are instrumented in the following components as the request, for example, transaction, travels through the Web server, Proxy Server and the application server:

- the Web server plug-in which is only available when using the Web server port.
- the Proxy Server, instrumented as servlet and Web services requests.
- the Web container, including servlet and servlet filters.
- the Enterprise JavaBeans (EJB) container.
- Java DataBase Connectivity (JDBC) calls.
- J2EE Connector Architecture (JCA).
- Web services, both on the server and the client side.
- the Java Message Service (JMS) engine.
- Service Integration Bus (SIB).
- Portlet container, including the portlet requests.
- Asynchronous Beans.

Select which components that you want to instrument. For example, if you want instrumentation data only for the Web container and the JMS API, select this data in the administrative console and the detailed instrumentation data is generated only for the components that you select. The edge transactions are traced for the other components that are not specified for instrumentation.

When filtering is enabled, only requests that match the filter generate request metrics data, create log records, or call the ARM interfaces. You can add work into a running system specifically to generate trace information to evaluate the performance of specific types of requests in the context of normal load, ignoring requests from other sources that might affect the system. If the request matches any filter with a trace level greater than None, trace records are generated for that request.

Getting performance data from request metrics

This topic describes how to enable request metrics.

About this task

Request metrics is a tool that enables you to track individual transactions, recording the processing time in each of the major WebSphere Application Server components.

You can enable request metrics from the following locations:

- The administrative console. To enable request metrics in the administrative console, refer to the instructions under the Steps for this task section that follows.
- Command line. Java Management Extensions (JMX) interfaces are exposed for enabling request metrics through external tools. For more details on the exposed interfaces, refer to the request metrics API documentation.

To enable request metrics in the administrative console:

1. Open the administrative console.
2. Click **Monitoring and Tuning** > **Request Metrics** in the console navigation tree.
3. Verify that **Prepare servers for request metrics collection** check box is selected. If **Prepare servers for request metrics collection** check box is not selected, you will have to select it, save the change, and restart the server.
4. Select All, None, or Custom under **Components to be instrumented** to specify the request metrics components. If you select **Custom**, be sure to select the components you would like to enable.
5. Specify the components that are instrumented by request metrics.
6. Specify how much data to collect.
7. Enable and disable logging.
8. Enable Application Response Measurement (ARM) Agent.
9. Specify which ARM type to use.
10. Specify the name of the ARM transaction factory implementation class.
11. Isolate performance for specific types of requests.
    a. Add and remove request metrics filters.
12. Click **Apply** or **OK**.
13. Click **Save**.

Results

The request metrics is enabled.

What to do next

To ensure that the Web server plug-in recognizes the changes you made for the request metrics configuration, follow the steps in “Regenerating the Web server plug-in configuration file” on page 263, if logging time spent in the Web server.

Request metrics

Use this page to enable request metrics, select the components that are instrumented by request metrics, set trace levels, enable standard logs, enable Application Response Measurement (ARM), specify the type of ARM agent, and specify the ARM transaction factory implementation class name.

To view this administrative console page, click **Monitoring and Tuning** > **Request Metrics**.

Prepare Servers for request metrics collection

Turns on the request metrics feature.

When unchecked, the request metrics function is disabled. To enable request metrics, check the box, save the change, and restart the server. When it is checked, the server is ready to enable request metrics and you will not need to restart the server. Depending on what option is selected under **Components to be instrumented**, request metrics instrumentation will be enabled/disabled in various components.

Note: This selection process differs from the **Enable Request Metrics** check box in WebSphere Application Server Version 6.0x. You now have the option of selecting All, None, or Custom. If you select **Custom**, you must specify which components you would like to enable.

Enable request metrics

This turns on the request metrics feature. When disabled, the request metrics function is disabled.

Components to be instrumented

Selects the components that are instrumented by request metrics.

When **None** is selected, no request metrics instrumentation will be enabled. When **All** is selected, request metrics instrumentation will be enabled in all the components listed under **Custom**. When **Custom** is selected, request metrics instrumentation will be enabled in the selected components.

Note: An edge transaction, which is defined as the first transaction that enters the application server without a parent correlator, will always be instrumented even if the corresponding component is disabled for instrumentation.

Trace level

Specifies how much trace data to accumulate for a given transaction. Note that **Trace level** and **Components to be instrumented** work together to control whether or not a request will be instrumented.

Including one of the following values:

None No instrumentation.

Hops Generates instrumentation information on process boundaries only. When this setting is selected, you see the data at the application server level, not the level of individual components such as enterprise beans or servlets.

Performance_debug
Generates the data at Hops level and the first level of the intra-process servlet and Enterprise JavaBeans (EJB) call (for example, when an inbound servlet forwards to a servlet and an inbound EJB calls another EJB). Other intra-process calls like naming and service integration bus (SIB) are not enabled at this level.

Debug
Provides detailed instrumentation data, including response times for all intra-process calls.

Note: Requests to servlet filters will only be instrumented at this level.

Standard logs
Enables the request metrics logging feature.

Select this check box to trigger the generation of request metrics logs in the `SystemOut.log` file.

Note: Since enabling the request metrics logging feature will increase processor usage, it is recommended using this feature together with filters so that only selected requests are instrumented.

Application Response Measurement (ARM) agent
Enables request metrics to call an underlying Application Response Measurement (ARM) agent.

Before enabling ARM, you need to install an ARM agent and configure it to the appropriate classpath and path, following the instructions of the ARM provider.

Specify ARM agent
Specifies the type of ARM agent that you want to use.

The ARM 4.0 agent and Tivoli ARM 2.0 agent are supported.

ARM transaction factory implementation class name
Specifies the ARM transaction factory implementation class name in the package that is supplied by your provider. This field is required when ARM 4.0 agent is selected, but it is not required when Tivoli ARM agent is selected.

In this field, type the name of the ARM transaction factory implementation class that is present in the ARM library. Be sure to follow the instructions of the ARM provider and understand the name of the ARM transaction factory class for the installed ARM agent.

Filters
When filtering is enabled, only requests matching the specified filter will generate request metrics data. Filters exist for source IP address, URI name, EJB method name, JMS parameters, and the WebServices parameters.

Application Response Measurement
Request metrics information might be either saved to the log file for later retrieval and analysis, be sent to Application Response Measurement (ARM) agents, or both. Request metrics provides response time for each of the major WebSphere Application Server components through ARM APIs.

ARM is an Open Group standard. Request metrics helps you to plug in an ARM agent to collect response time measurements.

WebSphere Application Server does not ship an ARM agent. However, it supports the use of agents adhering to ARM 4.0 and ARM 2.0 standards.

You can choose your own ARM implementation providers to obtain the ARM implementation libraries. Follow the instruction from the ARM provider, and ensure that the ARM API Java archive (JAR) files found in the ARM provider are on the class path so that the WebSphere Application Server can load the needed classes. In the case of Tivoli Monitoring Transaction Performance, V5.3, copy the `armjni.jar` and `core_util.jar` files from the Tivoli Monitoring Transaction Performance *<tmtp_install_root>*`/lib` installation root directory to the *app_server_root*`/lib` directory, which is the WebSphere Application Server installation root directory. If the underlying ARM implementation is ARM 4.0, you need to specify the ARM transaction factory class name. Otherwise, this specification is not required.

See the article “Performance: Resources for learning” on page 1 for more information about the ARM specifications.

ARM application properties and transaction context data

Request metrics provides build-in instrumentation to monitor transaction flows. The data that is collected by request metrics can be sent to a supported Application Response Measurement (ARM) agent.

ARM application properties

The following tables show the ARM application properties when request metrics initializes an ARM 4.0 agent.

Identity property names	Value
Cell Name	The cell name that the server belongs to.
Version	The version of WebSphere Application Server.

Application Properties	Value
Registered application name	WebSphere:<server_type> The <server_type> can be APPLICATION_SERVER, ONDEMAND_ROUTER, or PROXY_SERVER.
Application group	<server_name>
Application instance	<node_name>.<server_name>

The following code sample displays how WebSphere Application Server creates an ARM application.

```
String serverType; // the server type like APPLICATION_SERVER
String version; // WebSphere version
String sCellName; // the cell name of dMgr
String sAppInstance. // <short_node_name>.<short_server_name>
String sServerInstance; // short server name
String sWasName = "WebSphere:" + serverType;
String[] IDNAMES = new String[]{"Cell Name", "Version"};
ArmIdentityProperties appIdentity = txFactory.newArmIdentityProperties(IDNAMES, new String[]{sCellName,
version}, null);
ArmApplicationDefinition appDef = txFactory.newArmApplicationDefinition( sWasName, appIdentity, null );
ArmApplication app = txFactory.newArmApplication(appDef, sServerInstance, sAppInstance, null );
```

ARM transaction types

You can use the following code sample to create an instance of ARM transaction.

```
String[] contextNames; // the names for the context data like Port, QueryString, URI, EJBName
String tranIdentityName; // the transaction types shown in the following table like URI, EJB.
String appDef; // defined and created in above code snippet under ARM application properties
String app; // defined and created in above code snippet under ARM application properties
ArmIdentityPropertiesTransaction props = armFactory.newArmIdentityPropertiesTransaction( null, null,
contextNames, null );
ArmTransactionDefinition atd = armFactory.newArmTransactionDefinition(appDef, identityName, props, (ArmID)null);
ArmTransaction at = txFactory.newArmTransaction(app, atd );
```

The sections below show all ARM transaction types and their corresponding context names. Some types of transactions are not instrumented unless the trace level is set to DEBUG. In addition, not every transaction is available in all types of servers. All transaction types and context names are case sensitive.

- “Uniform resource identifier (URI)” on page 253
- “Enterprise JavaBeans (EJB)” on page 253
- “Servlet filter” on page 254
- “Java Database Connectivity (JDBC)” on page 254
- “Java Connector Architecture (JCA)” on page 254

Uniform resource identifier (URI)

This transaction type is the uniform resource identifier (URI) for servlet and JavaServer Page (JSP) requests. All of the following transaction types and context names are available in all types of servers.

Transaction type: URI	
Context name	**Description**
Port	The TCP/IP port where the request arrived, specified as a string representation of the decimal value. **Example:** 9080
QueryString	The portion of the dynamic URI that contains the search parameters of the request in the query segment of the URI string, excluding the question mark (?) character. **Example:** selection=EJB&lookup=GBL&trans=CMT
URI	The incoming request URI of the servlet and JSP file used. **Example:** /hitcount

Enterprise JavaBeans (EJB)

This EJB transaction type and context names are only available in application servers.

Transaction Type: EJB	
Context name	**Description**
EJBName	The fully qualified EJB class name, followed by method name with a period (.) to connect them. For example, com.mypackge.MyEJBClass.mymethod. **Example:** com.ibm.defaultapplication.IncrementBean.create
ApplicationName	The Java™ 2 Platform, Enterprise Edition (J2EE) application name that contains the enterprise bean. **Example:** DefaultApplication
ModuleName	The J2EE module name that contains the enterprise bean. **Example:** Increment.jar

Servlet filter

The servlet filter transaction type is only available at the DEBUG trace level. For other levels, servlet filters are not instrumented. This transaction type and context names are only available in application servers.

Transaction Type: Servlet filter	
Context name	**Description**
FilterName	The servlet filter name for the servlet. **Example:** ALoginFilter

Java Database Connectivity (JDBC)

The JDBC transaction type is only available at the DEBUG trace level. For other levels, JDBC is a block call. This transaction type and context names are only available in application servers.

Transaction type: JDBC	
Context name	**Description**
ClassName	The interface name for the JDBC call. This name is not the actual class name. **Example:** java.sql.PreparedStatement
MethodName	The method name for the JDBC call. **Example:** executeUpdate()

Java Connector Architecture (JCA)

The JCA transaction type is available only at the DEBUG trace level. For other levels, JCA is a block call. This transaction type and context names are only available in application servers.

Transaction type: JCA	
Context name	**Description**
ClassName	The class name for the JCA call. **Example:** javax.resource.spi.ManagedConnection
MethodName	The method name for the JCA call. **Example:** getConnection(Subject, ConnectionRequestInfo)

Web Services Provider

The Web services provider transaction type is for server-side Web services requests. This transaction type and the *Wsdlport* and *Operation* context names are available in all types of servers. The *Transport*, *NameSpace*, and *InputMessage* context names are only available in application servers.

Transaction type: Web Services Provider	
Context name	**Description**
WsdlPort	The WSDL port name that is associated with the Web service. **Example:** AccountManager

Transaction type: Web Services Provider	
Operation	The operation name that is associated with the Web service. **Example:** createNewAccount
Transport	The transport name that is associated with the Web service. **Example:** http
NameSpace	The name space that is associated with the Web service. **Example:** http://accountmanager.mycorp.com
InputMessage	The input message name that is associated with the Web service. **Example:** createNewAccountRequest

Web Services Requestor

The Web services requestor transaction type is available only at DEBUG trace level. For other levels, Web services requestor is a block call. This transaction type and context names are only available in application servers.

Transaction type: Web Services Requestor	
Context name	**Description**
WsdlPort	The Web Services Description Language (WSDL) port name that is associated with the Web service. **Example:** AccountManager
Operation	The operation name that is associated with the Web service. **Example:** createNewAccount
Transport	The transport name that is associated with the Web service. **Example:** http
Parameters	The parameters for the Web service request. **Example:** customerName,addressStreet,addressCity,addressState,addressZip

Java Message Service (JMS)

The JMS transaction type is only for message-driven bean (MDB) scenarios in default messaging, including the System Integration Bus (SIB) layer and MQ. This transaction type and context names are only available in application servers.

Transaction type: JMS	
Context name	**Description**
DestinationName	The destination queue name or topic name for the Java Message Service (JMS). **Example:** MyBusiness.Topic.Space

Transaction type: JMS	
MessageSelector	The message selector for JMS.
Provider	Provider refers to either default messaging, SIB, or MQ. **Example:** Default Messaging

JMS send and receive

The JMS send and receive transaction type is only available at the DEBUG trace level. For other levels, JMS send and receive is a block call. This transaction type and context names are only available in application servers.

Transaction type: JMS send/receive	
Context name	**Description**
ClassName	The class name in the JMS provider that makes the call. **Example:** com.ibm.ws.sib.api.jms.impl.JmsTopicPublisherImpl
MethodName	The method name in the JMS provider that makes the call. **Example:** sendMessage

Service Integration Bus (SIB) send and receive

The SIB send and receive transaction type is only available at the DEBUG trace level. For other levels, SIB send and receive is a block call. This transaction type and context names are only available in application servers.

Transaction type: SIB send/receive.	
Context name	**Description**
ClassName	The class name in the SIB layer that makes the call. **Example:** com.ibm.ws.sib.processor.impl.ProducerSessionImpl
MethodName	The method name in the SIB layer that makes the call. **Example:** send
BusName	The name of the Service Integration Bus that this call is made on.
DestinationName	The name of the Service Integration Destination that this call is made on.

SIB message-driven bean (MDB)

The SIB MDB transaction type is only available in application servers.

Transaction type: SIB MDB.	
Context name	**Description**
BusName	The name of the SIB that the Destination the MDB is listening on is located.

Transaction type: SIB MDB.	
DestinationName	The name of the SIB that this MDB is listening too.
MessageSelector	The message selector used by this MDB.
MdbDiscriminator	The Discriminator used by this MDB.
Provider	The type of provider that this MDB is for. **Example:** Default Messaging or SIB

SIB mediate

The SIB mediation transaction type and context names are only available in application servers.

Transaction type: SIB mediate	
Context name	**Description**
ClassName	The class name in which the call is made.
MethodName	The method name in which the call is made.
MediationName	The mediation name for the JMS. **Example:** myMediation
BusName	The service integration bus name for the JMS. **Example:** thisBusName
DestinationName	The destination name for the JMS. **Example:** myMessageQueue

Asynchronous beans

The asynchronous beans transaction type is for asynchronous beans timers, alerts, and deferred starts. This transaction type and context names are only available in application servers.

Transaction type: Asynchronous beans	
Context name	**Description**
Type	The type of asynchronous beans task. **Example:** COMMONJ_TIMER
ClassName	The class name that executes the asynchronous beans task. **Example:** com.mycorp.MyTaskClass

Java Naming and Directory Interface (JNDI)

The JNDI transaction type is only available at DEBUG trace level. For other trace levels, JNDI calls will not be instrumented. This transaction type and context names are only available in application servers.

Transaction type: JNDI	
Context name	**Description**
JNDIName	The JNDI lookup name. **Example:** ejbJndiName

Portlet

The portlet transaction type is and context names are only available in application servers.

Transaction type: Portlet	
Context name	**Description**
Method	The method of the portlet that is currently utilized, which can be either action or render.
WindowID	The window identifier of the portlet that is currently utilized.
URI (This is an ARM transaction context property name, and not to be confused with the explicit ARM transaction URI property as specified on WebSphere servlet transactions, and in particular not to be confused with the Enterprise Workload Manager (EWLM) ″EWLM: URI″ policy filter.)	The context path of the current portlet request.

ARM correlators via HTTP and SOAP protocols

For HTTP inbound, WebSphere Application Server checks the case sensitive HTTP header “ARM_CORRELATOR” for the incoming ARM correlator. The application server does not allow the flow of ARM correlators via HTTP outbound.

For SOAP inbound, WebSphere Application Server checks the SOAP header for an element as follows:

- element =″arm_correlator″;
- namespace URI = “http://websphere.ibm.com”,
- prefix = ″reqmetrics″;
- actor URI = ″reqmetricsURI″;

For SOAP outbound, WebSphere Application Server creates a new SOAP header and passes the ARM correlator string as a text node in the header. For example:

- element =″arm_correlator″;
- namespace URI = “http://websphere.ibm.com”,
- prefix = ″reqmetrics″;
- actor URI = ″reqmetricsURI″;

The following is an example SOAP header with an ARM correlator. Note that the ARM correlator must be passed in hex string format of the ARM correlator byte array despite whether HTTP or SOAP protocol is used.

```
<soapenv:Header xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<reqmetrics:arm_correlator soapenv:actor="reqmetricsURI" xmlns:reqmetrics="http://websphere.ibm.com">
37000000000000000000
</reqmetrics:arm_correlator>
</soapenv:Header>
```

Isolating performance for specific types of requests

This topic describes how to enable request metrics filters.

About this task

Request metrics compares each incoming request to a set of known filters, but you need to enable these filters.

1. Open the administrative console.
2. Click **Monitoring and Tuning** > **Request Metrics** in the administrative console navigation tree.
3. Click **Filters**.
4. Click *filter type*.
5. Select the check box in the **Enable** field under the Configuration tab.
6. Click **Apply** or **OK**.
7. Click **Save**. You can enable or disable a filter group. If the group is enabled, you can enable or disable individual filters.

Results

The request metrics filters are enabled according to your configuration. For example, if you enabled source IP, only requests whose source IP matches the one specified in the filter will be instrumented.

Note: Filters will only be checked for edge transactions. An edge transaction is the transaction that first enters an instrumented system. For example, if a servlet calls an Enterprise JavaBean, the servlet is the edge transaction, assuming it is not instrumented at the Web server plug-in, and the URI and SOURCE_IP filters will be checked for the servlet request. However, when the request comes to EJB container, EJB filter will not be checked because it is no longer an edge transaction.

What to do next

You must regenerate the Web server plug-in configuration file as described in the “Regenerating the Web server plug-in configuration file” on page 263 topic after modifying the request metrics configuration.

Adding and removing request metrics filters

This topic summarizes how to add and remove request metrics filter.

About this task

To add or remove request metrics filters, perform the following steps:

1. Open the administrative console.
2. Click **Monitoring and Tuning** > **Request Metrics** in the console navigation tree.
3. Click **Filters**.
4. Choose a filter type.
   a. Click **Filter values**.
   b. You can edit, add, and delete a filter value. To edit, click a filter value and change its value. To add, click **New** and type in the value and optionally check the **Enable filter** box. To delete, select a filter value and click **Delete**.
5. Click **Apply** or **OK**.
6. Click **Save**.

Results

Adding or removing request metrics filters is complete.

What to do next

You must regenerate the Web server plug-in configuration file as described in the “Regenerating the Web server plug-in configuration file” on page 263 topic after modifying the request metrics configuration.

Request metrics filters

Use this page to view a list of request metrics filters.

To view this administrative console page, click **Monitoring and Tuning** > **Request Metrics** > **Filters**.

Maximum rows: Specifies the maximum number of rows that displays when the collection is large. The rows that are not displayed appear on the next page.

Retain filter criteria: Specifies whether to use the same filter criteria entered in the show filter function to display this page the next time you visit it.

Type:

Specifies the type of request metrics filter.

Enable:

Specifies whether this filter is enabled. This option must be enabled to enable the filter values under this filter type.

Request metrics filter settings

Use this page to specify filters that define whether or not trace is enabled for inbound requests. Note that the filters will only be checked for inbound requests. For intra-process and outbound calls, filters will not be checked. If an inbound request passes the relevant filters, the request will be instrumented as it moves through WebSphere Application Server. If an inbound request does not pass the relevant filters, the request will not be instrumented for its entire code path in WebSphere Application Server.

To view this administrative console page, click **Monitoring and Tuning** > **Request Metrics** > **Filters** > *filter_type*.

Type:

Specifies the type of request metrics filter.

Enable:

Specifies whether this filter is enabled.

Filter values:

Specifies the value(s) of the request metrics filter type and enablement for the value(s).

Filter values collection

Use this page to specify the values for source IP, URI, Web services, Java Message Service (JMS), or Enterprise JavaBeans (EJB) request metrics filters. When multiple filter values are enabled, a request will pass the filter as long as it matches one of the filter values.

To view this administrative console page, click **Monitoring and Tuning** > **Request Metrics** > **Filters** > *filter_type* > **Filter values**.

Maximum rows: Specifies the maximum number of rows that displays when the collection is large. The rows that are not displayed appear on the next page.

Retain filter criteria: Specifies whether to use the same filter criteria entered in the show filter function to display this page the next time you visit it.

Value:

Specifies a source IP, URI, Web services, JMS, or EJB value based on the type of filter.

For example, for URI filters, the value might be ″/servlet/snoop″.

Enable filter:

Specifies whether a filter value is enabled.

Filter values settings

Use this page to specify the values for source IP, URI, Web services, Java Message Service (JMS), or Enterprise JavaBeans (EJB) 2method name request metrics filters.

To view this administrative console page, click **Monitoring and Tuning** > **Request Metrics** > **Filters** > *filter_type* > **Filter values** > *filter_value*.

Value:

Specifies a source IP, URI, Web services, JMS, or EJB value based on the type of filter.

For example, for URI filters, the value can be `/servlet/snoop`. When needed, the wildcard (*) can be appended at the end of the value. For example, `/servlet/s*` is a valid URI filter value.

For JMS and Web services filter values, the wildcard can be used at the end of one or more name/value pairs.

Enable filter:

Specifies whether this filter value is enabled.

Specifying how much data to collect

This topic describes how to set the trace level to generate trace records in the administrative console.

About this task

Trace level specifies how much trace data to accumulate for a given transaction. You should choose the one of the following values when setting the trace level in the administrative console.

None No instrumentation.

Hops

Performance_debug

Debug

1. Open the administrative console.
2. Click **Monitoring and Tuning** > **Request Metrics** in the administrative console navigation tree.
3. Find **Trace level** in the Configuration tab.
4. Select a trace level from the drop down list box. To set the request metrics trace level to generate records, make sure that the trace level is set to a value greater than `None`.
5. Click **Apply** or **OK**.
6. Click **Save**.

Results

Information are generated to reflect the trace level you selected.

What to do next

You must regenerate the Web server plug-in configuration file as described in the “Regenerating the Web server plug-in configuration file” on page 263 topic after modifying the request metrics configuration.

Request metrics trace filters

When request metrics is active, trace filters control which requests get traced. The data is recorded to the system log file or sent through Application Response Measurement (ARM) for real-time and historical analysis.

Incoming HTTP requests

HTTP requests that arrive at WebSphere Application Server might be filtered based on the URI or the IP address or both of the originator of the request.

.

- **Source IP address filters.** Requests are filtered based on a known IP address. You can specify a mask for an IP address using the asterisk (*). If used, the asterisk must always be the last character of the mask, for example 127.0.0.*, 127.0.*, 127*. For performance reasons, the pattern matches character by character, until either an asterisk is found in the filter, a mismatch occurs, or the filters are found to be an exact match.

  Only addresses that are entered in one of the following addressing formats are acceptable as the source IP addresses:
  - The IPv4 addressing format:

    ```
    ^(\d{1,2}|1\d\d|2[0-4]\d|25[0-5])\.(\d{1,2}|1\d\d|2[0-4]\d|25[0-5])\.
    (\d{1,2}|1\d\d|2[0-4]\d|25[0-5])\.(\d{1,2}|1\d\d|2[0-4]\d|25[0-5])$
    ```
  - The IPv6 addressing format:

    ```
    ^([0-9a-fA-F]*[0-9a-fA-F]*[0-9a-fA-F]*[0-9a-fA-F]*:){1,7}([0-9a-fA-F]*
    [0-9a-fA-F]*[0-9a-fA-F]*[0-9a-fA-F]*)$
    ```
  - The IPv4 compatiable IPv4 addressing format:

    ```
    ^([0]*[0]*[0]*[0]*:){1,7}((\d{1,2}|1\d\d|2[0-4]\d|25[0-5])\.
    (\d{1,2}|1\d\d|2[0-4]\d|25[0-5])\.(\d{1,2}|1\d\d|2[0-4]\d|25[0-5])\.
    (\d{1,2}|1\d\d|2[0-4]\d|25[0-5]))$/<constant-value>
    ```
  - The IPv4MappedIPv6 addressing format:

    ```
    ^([0]*[0]*[0]*[0]*:){1,6}([fF][fF][fF][fF]:){1}((\d{1,2}|1\d\d|2[0-4]
    \d|25[0-5])\.(\d{1,2}|1\d\d|2[0-4]\d|25[0-5])\.(\d{1,2}|1\d\d|2[0-4]\d|25[0-5])\.
    (\d{1,2}|1\d\d|2[0-4]\d|25[0-5]))$
    ```

  Note: If the client machine is a dual stack machine and its IP address is specified as the source IP address that is filtered by the request metrics filter, you must specify the IPv4 addressing format rather than the IPv6 addressing format. Only if the client machine is a single stack IPv6 machine, can you specify it as the IPv6 addressing format
- **URI filters.** Requests are filtered, based on the URI of the incoming HTTP requests. The rules for pattern matching are the same as for matching source IP address filters.
- **Filter combinations.** If both URI and source IP address filters are active, request metrics requires a match for both filter types. If neither is active, all requests are considered a match.

Incoming enterprise bean requests

Incoming enterprise bean requests

- Requests are filtered based on the full name of the EJB method. As with IP address and URI filters, the asterisk (*) might be used in the mask. If used, the asterisk must always be the last character of a filter pattern.

Because the ability to track the request response times comes with a cost, filtering helps optimize performance when using request metrics.

The Web services filter and the Java Message Service (JMS) filter was added to the WebSphere Application Server, Version 6 product. The filter values for Web services are a combination of a Web Services Description Language (WSDL) port name, operation name, and transport name. The filter value for JMS is the destination name.

Related tasks

"Specifying how much data to collect" on page 261
This topic describes how to set the trace level to generate trace records in the administrative console.

Regenerating the Web server plug-in configuration file

This topic describes the steps to regenerate the Web server plug-in configuration file after you modify the request metrics configuration.

About this task

After modifying the request metrics configuration, you must complete the following steps to regenerate the Web server plug-in configuration file. Regeneration ensures that the Web server plug-in recognizes the changes that you made for the request metrics configuration. If you make multiple changes to request metrics, then regenerate the plug-in configuration files when you complete all the changes.

Important: You must complete this step after you change the request metrics configuration. If you do not, the Web server plug-in might have different request metrics configuration data than the application server. This difference in configuration data might cause inconsistent behaviors for request metrics between the Web server plug-in and the application server.

1. Open the administrative console.
2. Click **Servers > Server Types > Web Servers** .
3. Select one of the listed Web servers, and click **Generate Plug-in**

Results

The regeneration of the Web server plug-in configuration file is complete.

Enabling and disabling logging

This topic describes how to enable and disable logging through the administrative console.

About this task

You can enable and disable logging through the administrative console to start the generation of request metrics logs in the SystemOut.log file in the following directory:

profile_root/logs/*server_name*

1. Open the administrative console.
2. Click **Monitoring and Tuning > Request Metrics**.
3. Under **Request Metrics Destination**, select the **Standard Logs** check box to enable the logging feature. To disable the logging , clear the **Standard Logs** check box.

Results

The logging feature is enabled. You can view the generated request metrics logs in the SystemOut.log file.

Request metrics performance data

Use this page to learn how to interpret performance data for request metrics in trace record format.

The trace records for request metrics data are output to two log files: the Web server plug-in log file and the application server log file. The default names for the log files are SystemOut.log and http_plugin.log. You might, however, specify these log file names and their locations. The default directories for these log files are:

- *plugin_install_root*/logs/*web_server_name*/http_plugin.log and *install_root*/profiles/*profile_name*/logs/*server_name*

and

In the WebSphere Application Server log file the trace record format is:

```
PMRM0003I: parent:ver=n,ip=n.n.n.n,time=nnnnnnnnnn,pid=nnnn,reqid=nnnnnn,event=nnnn
-
current:ver=n,ip=n.n.n.n,time=nnnnnnnnnn,pid=nnnn,reqid=nnnnnn,event=nnnn
           type=TTT detail=some_detail_information elapsed=nnnn
```

In the Web server plug-in log file the trace record format is:

```
PLUGIN:
parent:ver=n,ip=n.n.n.n,time=nnnnnnnnnn,pid=nnnn,reqid=nnnnnn,event=nnnn
- current:ver=n,ip=n.n.n.n,time=nnnnnnnnnn,pid=nnnn,reqid=nnnnnn,event=nnnn
           type=TTT detail=some_detail_information elapsed=nnnn bytesIn=nnnn
           bytesOut=nnnn
```

The trace record format is composed of two correlators: a parent correlator and current correlator. The parent correlator represents the upstream request and the current correlator represents the current operation. If the parent and current correlators are the same, then the record represents an operation that occurs as it enters WebSphere Application Server.

To correlate trace records for a particular request, collect records with a message ID of PMRM0003I from the appropriate application server log files and the PLUGIN trace record from the Web server plug-in log file. Records are correlated by matching current correlators to parent correlators. You can create the logical tree by connecting the current correlators of parent trace records to the parent correlators of child records. This tree shows the progression of the request across the server cluster. Refer to “Why use request metrics?” on page 245 for an example of the transaction flow.

The parent correlator is denoted by the comma separating fields following the keyword, parent:. Likewise, the current correlator is denoted by the comma separating fields following, current:.

The fields of both parent and current correlators are:

- **ver:** The version of the correlator. For convenience, it is duplicated in both the parent and current correlators.
- **ip:** The IP address of the node of the application server that generated the correlator. If the system has multiple IP addresses, request metrics uses one of the IP addresses to identify the system.
- **pid:** The process ID of the application server that generated the correlator.
- **time:** The start time of the application server process that generated the correlator.
- **reqid:** An ID that is assigned to the request by request metrics, unique to the application server process.
- **event:** An event ID that is assigned to differentiate the actual trace events.

Following the parent and current correlators, the metrics data for timed operation are:

- **type**: A code that is represents the type of operation being timed. Supported types include HTTP, URI, EJB, JDBC, JMS, COMMONJ_WORK_POOLED, COMMONJ_TIMER, Web services requester, and Web services provider.
- **detail**: Identifies the name of the operation being timed (See the following description of Universal Resource Identifier (URI), HTTP, EJB, JDBC, JMS, asynchronous beans, and Web services.)
- **elapsed**: The measured elapsed time in <units> for this operation, which includes all sub-operations called by this operation. The unit of elapsed time is milliseconds.
- **bytesIn**: The number of bytes from the request that is received by the Web server plug-in.

- **bytesOut**: The number of bytes from the reply that is sent from the Web server plug-in to the client.

The type and detail fields that are described include:
- **HTTP**: The Web server plug-in generates the trace record. The detail is the name of the URI that is used to invoke the request.
- **URI**: The trace record is generated by a Web component. The URI is the name of the URI that is used to invoke the request.
- **EJB**: The fully qualified package and the method name of the enterprise bean.
- **JDBC** The interface name and method name for that JDBC call.
- **JMS:** JMS includes the particulars of various JMS parameters
- **Asynchronous beans:**The detail specifies the name of the asynchronous beans. Asynchronous beans include two types: `COMMONJ_WORK_POOLED` and `COMMONJ_TIMER`.
- **Web services:**Web services include the particulars of various Web services parameters. Web services include two types: `Web services requestor` and `Web services provider`.
- **SIB**: Used for instrumentation in service integration bus including message send/receive and mediation.
- **JCA**: J2EE Connector Architecture. The detail specifies the class name in which the JCA call is made.
- **JNDI**: Used for JNDI naming look up. The detail specifies the JNDI name.
- **JMS send and receive:** Generates the trace record by JMS sending and receiving messages.
- **SIB send and receive:** Generates the trace record by SIB sending and receiving messages.

When there are multiple servant regions for an application server, there are multiple `SystemOut.log` files, one for each servant region. Therefore, request metrics might log the trace records in multiple `SystemOut.log` files. The servant region that handles a request logs the relevant records in its `SystemOut.log` files. The pid in the current request metrics correlator is the pid for the corresponding servant region. If the system has multiple IP addresses, the IP in the correlator could be one of them, but it should use the same IP for the same servant region.

Related tasks

"Enabling and disabling logging" on page 263
This topic describes how to enable and disable logging through the administrative console.

Request metric extension

Certain applications might require additional instrumentation points within the request metrics flow. For example, you might want to understand the response time to a unique back-end system as seen in the following call graph:

```
HTTP request /trade/scenario             ------------------------------> 172 ms
     Servlet/trade/scenario          -------------------------------> 130 ms
       Servlet/call to unique back-end system -------------------------->38 ms
```

Request metrics uses a *token* or *correlator* when tracing the flow of each request through the system. To create the call graph above with this instrumentation, you must plug into that flow of the request and issue the appropriate Application Response Measurement (ARM) API for an ARM agent to collect the data and for the ARM vendor to create the call graph.

Request metrics exposes the Correlation Service API for you to plug into the flow of the request. The following example is one of the typical flows that might be followed by an instrumented application to plug into the request metrics flow:

1. Create a new ArmTransaction object, which runs various instrumentation calls such as start or stop. The Correlation Service Arm wrapper (PmiRmArmTx) encapsulates this object before being inserted into the request metrics flow.
2. Populate the ArmTransaction object with an appropriate ArmCorrelator object. This object encapsulates the actual ARM correlator bytes.
3. Run the start method on the ArmTransaction object, marking the beginning of the instrumented method.

4. Instantiate a PmiRmArmTx object using the static method on the PmiRmArmTxFactory class, and populate it with the ArmTransaction object above.
5. Pass the PmiRmArmTx object above to the Correlation Service by pushing it onto the Correlation Service stack using exposed methods on the PmiRmArmStack class.
6. Perform the tasks that need to be done by the method being instrumented. The Correlation Service takes care of flowing the ArmTransaction object as necessary, which eventually results in the call graph view of the transaction times.
7. At the end of the instrumented method, access the PmiRmArmTx object from the Correlation Service using exposed methods on the PmiRmArmStack class, access the ArmTransaction object and perform a stop to indicate the end of the transaction.

Example: Using the correlation service interface

Consult this information when utilizing the ARM API with the correlation service as part of a servlet instrumentation.

The arm40 binaries should be installed in accordance with the installation instructions supplied by the implementation provider. Once this is done, restart the server. This causes trace records to be generated in the SystemOut.log file indicating the instantiation of the appropriate ARM implementation.

The following example illustrates one of the typical workflows of using the ARM API in conjunction with the correlation service as part of a servlet instrumentation:

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
  throws ServletException, IOException {

PmiRmArmTx artrax =
// The factory detects the currently active ARM implementation (specified by user through
// admin console) and instantiates an appropriate ARM wrapper object

    PmiRmArmTxFactory.createPmiRmArmTx();
ArmTransaction at = newArmTx();
if (null == at)
            out.println("Got a null ArmTransaction");
ArmCorrelator arc = newArmCorr();
at.start(arc);
try {
 artrax.setArmTransaction(at);
 PmiRmArmStack.pushTransaction(artrax);
} catch (Exception e) {
 System.out.println("Caught 1 exception" + e);
}

PmiRmArmTx atxwrp = PmiRmArmStack.peekTransaction();

if (atxwrp == null)
 out.println("Armtransaction is null");

//getArmType
try {
 out.println("ARMTYPE is"+ PmiRmArmTx.getARMType());
} catch (Exception e) {
 out.println(e);
}
//getting correlator bytes
try {
 if (null == atxwrp.getCorrelatorBytes())
  out.println("Got a null Correlator");
} catch (Exception e) {
 out.println(e);
}

//blocked/unblocked
```

```
long blkid = 0;
try {
 out.println(blkid = atxwrp.blocked());
} catch (Exception e) {
 out.println(e);
}

try {
 out.println(atxwrp.unblocked(blkid));
} catch (Exception e) {
 out.println(e);
}
try {
 atxwrp = PmiRmArmStack.popTransaction();
 ArmTransaction art = (ArmTransaction) atxwrp.getArmTransaction();
 art.stop(ArmConstants.STATUS_GOOD);
} catch (Exception e) {
 out.println(e);
}

}



private ArmTransaction newArmTx() {

 ArmTransactionFactory txFactory = null;
try {
 String sWasName = "WebSphere";
 String appName = "t23xpimage/t23xpimage/server1";
 String sCellName = appName.substring(0, appName.indexOf("/"));
 String sNodeInstance =
   appName.substring(appName.indexOf("/") + 1, appName.length());
 sNodeInstance = sNodeInstance.replace('/', '.');
 txFactory = (ArmTransactionFactory)
 newObjectInstance("org.opengroup.arm40.sdk.ArmTransactionFactoryImpl");
 ArmApplication app = null; // 149297
 ArmApplicationDefinition appDef = null; //LIDB3207
 appDef =    txFactory.newArmApplicationDefinition(sWasName, null, null);
 app = txFactory.newArmApplication(appDef, sCellName, sNodeInstance, null);

 String[] idnames = { "request_type" };
 String[] idvalues = { "URI" };
 String[] ctxnames = { "URI" };
 ArmIdentityPropertiesTransaction props =
   txFactory.newArmIdentityPropertiesTransaction(
     idnames,
     idvalues,
     ctxnames,
     null);
 ArmTransactionDefinition atd =
    txFactory.newArmTransactionDefinition(
     appDef,
     "URI",
     props,
     (ArmID) null);
 ArmTransaction at = txFactory.newArmTransaction(app, atd);
 return at;
} catch (Exception e) {
 System.out.println(e);
 return null;
}

}

private ArmCorrelator newArmCorr() {
```

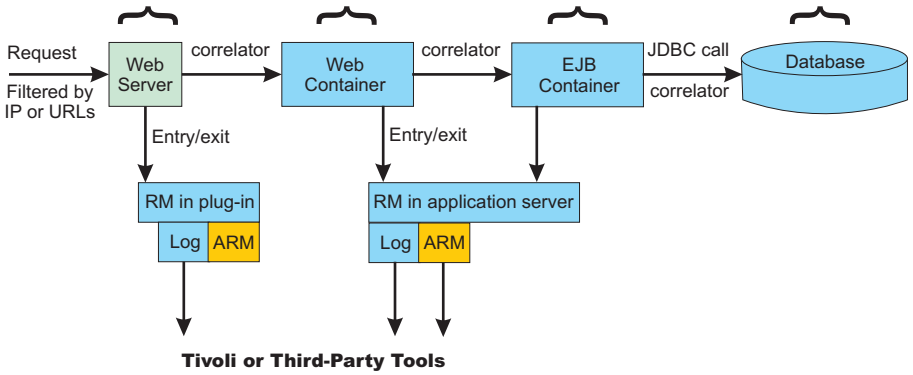
```
 ArmTransactionFactory txFactory = null;
try {
 String sWasName = "WebSphere";
 String appName = "t23xpimage/t23xpimage/server1";
 txFactory =
 (ArmTransactionFactory) newObjectInstance("org.opengroup.arm40.sdk.ArmTransactionFactoryImpl");

 ArmCorrelator arc =txFactory.newArmCorrelator(
     PmiRmArmStack.peekTransaction().getCorrelatorBytes());
 return arc;
} catch (Exception e) {
 System.out.println(e);
 return null;
}

}
```

There are several potential scenarios for using the PmiRmArmStack. This example shows a scenario where code accesses an existing PmiRmArmTx on the stack, extracts the correlator, and calls blocked and unblocked. This is a typical scenario when sending a correlator along an unsupported protocol. In this scenario, the Arm transaction is already on the stack.

```
1   PmiRmArmTx artrax =
2    PmiRmArmStack.peekTransaction();
3   if( artrax != null )
4                         {
5                                    try
6                                    {
7        byte[] cbytes = artrax.getCorrelatorBytes();
8                                       stuffBytesIntoOutboundMessage( msg, cbytes);
9                                       long blockedId = 0;
10                                      try
11                                      {
12                                         blockedId = artrax.blocked();
13                                       }
14                                       catch( NoSuchMethodException nsme )
15                                       {
16                                           // must not be running ARM4 or eWLM
17                                       }
18                                         sendMsg( msg );
19                                      try
20                                      {
21                                         artrax.blocked( blockedId );
22                                       }
23                                       catch( NoSuchMethodException nsme )
24                                       {
25                                           // must not be running ARM4 or eWLM
26                                       }
27
28                                   }
29                                   catch( Exception e )
30                                   {
31                                      report a problem;
32                                   }
33   }
```

Differences between Performance Monitoring Infrastructure and request metrics

Performance Monitoring Infrastructure (PMI) provides information about average system resource usage statistics, with no correlation between the data across different WebSphere Application Server components. For example, PMI provides information about average thread pool usage. Request metrics provides data about each individual transaction, correlating this information across the various WebSphere Application Server components to provide an end-to-end picture of the transaction, as shown in the following diagram:



Appendix. Directory conventions

References in product information to *app_server_root*, *profile_root*, and other directories infer specific default directory locations. This topic describes the conventions in use for WebSphere Application Server.

Default product locations - z/OS

app_server_root
Refers to the top directory for a WebSphere Application Server node.

The node may be of any type—application server, deployment manager, or unmanaged for example. Each node has its own *app_server_root*. Corresponding product variables are was.install.root and WAS_HOME.

The default varies based on node type. Common defaults are *configuration_root*/AppServer and *configuration_root*/DeploymentManager.

configuration_root
Refers to the mount point for the configuration file system (formerly, the configuration HFS) in WebSphere Application Server for z/OS.

The *configuration_root* contains the various *app_server_root* directories and certain symbolic links associated with them. Each different node type under the *configuration_root* requires its own cataloged procedures under z/OS.

The default is /wasv7config/*cell_name*/*node_name*.

plug-ins_root
Refers to the installation root directory for Web Server plug-ins.

profile_root
Refers to the home directory for a particular instantiated WebSphere Application Server profile.

Corresponding product variables are server.root and user.install.root.

In general, this is the same as *app_server_root*/profiles/*profile_name*. On z/OS, this will be always be *app_server_root*/profiles/default because only the profile name ″default″ is used in WebSphere Application Server for z/OS.

smpe_root
Refers to the root directory for product code installed with SMP/E.

The corresponding product variable is smpe.install.root.

The default is /usr/lpp/zWebSphere/V7R0.

Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, is the user's responsibility.

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

IBM Director of Intellectual Property & Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
USA

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

IBM Corporation
Mail Station P300
2455 South Road
Poughkeepsie, NY 12601-5400
USA
Attention: Information Requests

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

Trademarks and service marks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. For a current list of IBM trademarks, visit the IBM Copyright and trademark information Web site (www.ibm.com/legal/copytrade.shtml).

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Java is a trademark of Sun Microsystems, Inc. in the United States, other countries, or both.

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