WebSphere® Business Integration Server Foundation for z/OS V5.1

Performance Monitoring and Tuning

SA22-7982-00

Note

Before using this information, be sure to read the general information under “Notices” on page 133.

Compilation date: May 23, 2004

Contents

How to send your comments

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

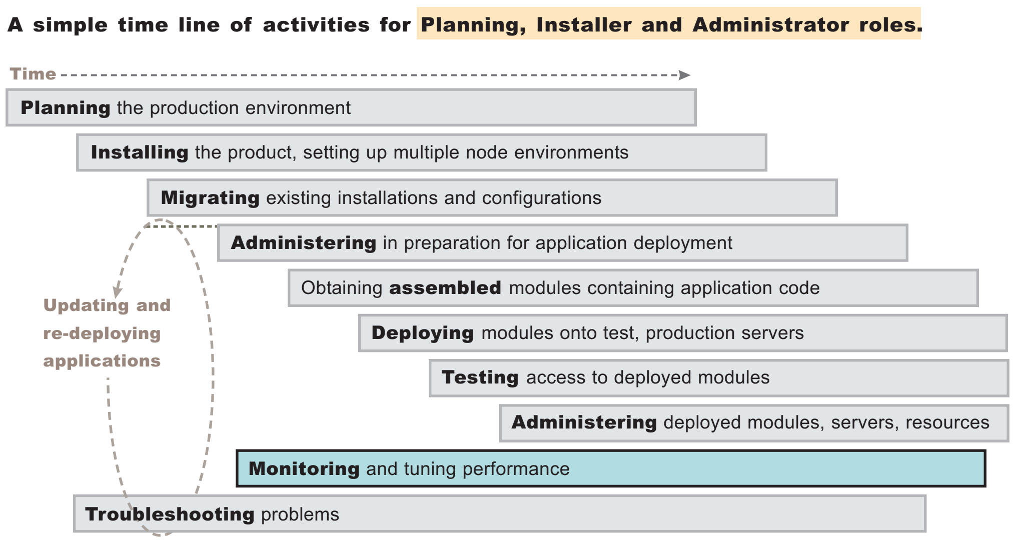
- To send comments on articles in the WebSphere Application Server Information Center
  1. Display the article in your Web browser and scroll to the end of the article.
  2. Click on the **Feedback** link at the bottom of the article, and a separate window containing an e-mail form appears.
  3. Fill 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 919-254-0206.

  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.

Chapter 1. Welcome to Monitoring

Performance monitoring is an activity in which you collect and analyze data about the performance of your applications and their environment.


A simple time line of activities for Planning, Installer and Administrator roles.
Time
Planning the production environment
Installing the product, setting up multiple node environments
Migrating existing installations and configurations
Administering in preparation for application deployment
Obtaining assembled modules containing application code
Deploying modules onto test, production servers
Testing access to deployed modules
Administering deployed modules, servers, resources
Monitoring and tuning performance
Troubleshooting problems
Updating and re-deploying applications

The product provides APIs and graphical interfaces for collecting and analyzing trace and log data.

Performance Monitoring Infrastructure (PMI)

The product collects data on run-time and applications through the Performance Monitoring Infrastructure (PMI), as described in “Performance Monitoring Infrastructure” on page 4. This infrastructure is compatible with and extends the JSR-077 specification.

PMI provides several types of interfaces to access performance data. Use these PMI interfaces to create tools that help monitor and tune performance.

Performance data can be monitored and analyzed with:
- Other Tivoli monitoring tools
- User-developed monitoring tools
- Third-party monitoring tools

The Tivoli performance viewer uses the PMI Java client to provide graphical displays and summary reports of collected data. For more information, see “Monitoring performance with Tivoli Performance Viewer (formerly Resource Analyzer)” on page 26.

Performance Monitoring Infrastructure (PMI) request metrics

Chapter 2. Monitoring performance

WebSphere for z/OS always collects certain performance data on z/OS which is available through the facilities of z/OS operating system which includes RMF data. Workload Management (WLM) delay monitoring is always enabled for WebSphere for z/OS. The Performance Monitoring Infrastructure (PMI) data is a non-platform specific WebSphere set of API's to do performance monitoring. The System Management facilities (SMF) interface is a z/OS platform way of collecting performance and accounting information. The PMI and SMF data are each individually configurable and need to be enabled. WebSphere for z/OS relies on its use of WLM services to collect some of the accounting and performance data. This information gets presented back to the installation through RMF and RMF-written SMF records. See Collecting job-related information with Systems Management Facility (SMF).

WebSphere Application Server collects data on run time and applications through the Performance Monitoring Infrastructure (PMI). Performance data can then be monitored and analyzed with a variety of tools.

To set up performance monitoring:

1. Setup workload management as described in Setting up workload management (WLM). Also see "Workload management (WLM) tuning tips for z/OS" on page 100.
2. Enable PMI services in the application server as described in "Enabling performance monitoring services in the application server through the administrative console" on page 21. To monitor performance data through the PMI interfaces, you must first enable the performance monitoring service through the administrative console before restarting the server.
3. Collect the data.

   The monitoring levels that determine which data counters are enabled can be set dynamically, without restarting the server. You can set these values in one of the following ways:

   a. "Starting the Tivoli Performance Viewer" on page 31.
   b. "Enabling performance monitoring services using the command line" on page 22.

   For WebSphere for z/OS, also refer to Collecting performance diagnosis information.

Performance information and accounting for WebSphere for z/OS

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

This information gets presented back to the installation through RMF and RMF-written SMF records. In addition, WebSphere for z/OS has its own SMF records which collect additional domain-specific information for WebSphere for z/OS. First, unless you need the SMF records or RMF data, turn them off using the Administration application and the SMFPRMxx parmlib statement. Refer to Setting up SMF and Enabling SMF recording. You also use the SMFPRMxx parmlib statement to control the detail of the WebSphere for z/OS SMF records. If you do need SMF information, you should review the SMF parmlib to ensure you are collecting only the data that you need (both record types and detail).

Setting up your workload manager goals and filtering criteria is probably beyond the scope of this section. However, you should recognize that you can classify work into service classes based on userid, server name, server instance name, and transaction class. Refer to Classifying WebSphere transaction workload for WLM. You should ensure that you classify the control regions as reasonably high-performing system tasks. Refer to “Workload management (WLM) tuning tips for z/OS” on page 100. Classification can be done in many ways now. servername and user is what is currently available for EJB’s. MDB’s and serlevt/jsp have there own mechanism for doing this which we need to ensure is documented.

Performance Monitoring Infrastructure

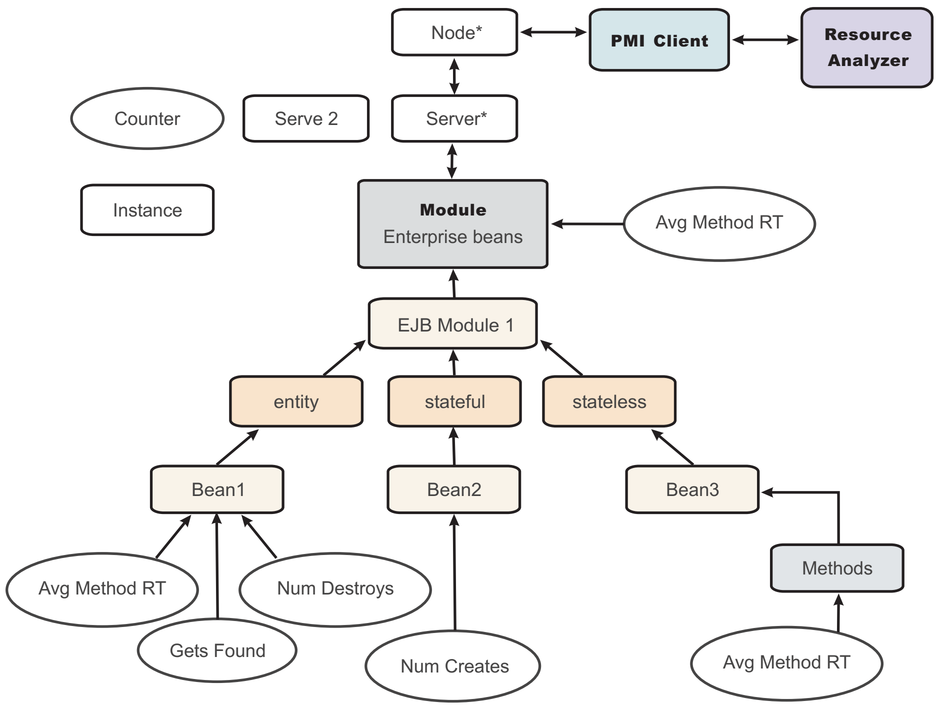
The Performance Monitoring Infrastructure (PMI) uses a client-server architecture. The server collects performance data from various WebSphere Application Server components. A client retrieves performance data from one or more servers and processes the data. Refer to “Performance Monitoring Infrastructure client interface” on page 38.

The server collects PMI data in memory. This data consists of counters such as servlet response time and data connection pool usage. The data points are then retrieved using a Web client, a Java client, or a Java Management Extensions (JMX) client. WebSphere Application Server contains Tivoli Performance Viewer, a Java client which displays and monitors performance data. See the “Monitoring performance with Tivoli Performance Viewer (formerly Resource Analyzer)” on page 26, “Third-party performance monitoring and management solutions” on page 72, and “Developing your own monitoring applications” on page 37 topics for more information on monitoring tools.

Performance data organization

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 Servers. Each server organizes PMI data into modules and submodules.

Hierarchy of data collections used for performance reporting to Resource Analyzer



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

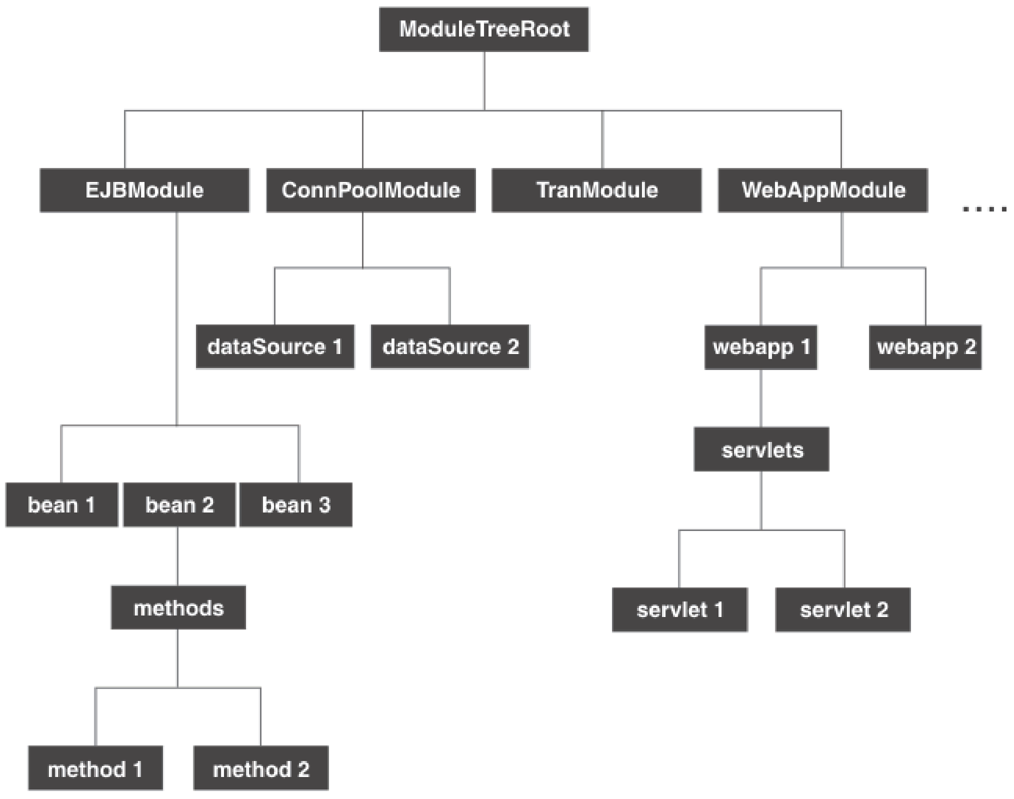
- **Node.** A node represents a physical machine in the WebSphere Application Server administrative domain.
- **Server.** A server is a functional unit that provides services to clients over a network. No performance data is collected for the server itself.
- **Module.** A module represents one of the 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 Profiler Interface (JVMPI)), 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.

The Tivoli Performance Viewer 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. For example, 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.

Each counter has a specified monitoring level: none, low, medium, high or maximum. If the module is set to lower monitoring level than required by a particular counter, that counter is not enabled. Thus, if Bean1 has a medium monitoring level, Gets Found and Num Destroys are enabled because they require a low monitoring level. However, Avg Method RT is not enabled because it requires a high monitoring level.

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.

JDBC connection pools
Data counters for this category contain usage information about connection pools for a database. Examples include the average size of the connection pool or number of connections, the average number of threads waiting for a connection, the average wait time in milliseconds for a connection, and the average time the connection is in use.

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.

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.

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.

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.

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 Java Bean (EJB) module PMI data (BeanModule)
- EntityBean*: specific EJB PMI data (BeanModule)
- JDBCProvider*: Java Database Connectivity (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 37.

Enterprise Java Bean counters

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

Counter definitions

Name	Key	Description	Version	Granularity	Type	Level
Num creates	beanModule.creates	The number of times that beans were created	3.5.5 and later	Per home	CountStatistic	Low
Num removes	beanModule.removes	The number of times that beans were removed	3.5.5 and later	Per home	CountStatistic	Low

Num passivates	beanModule.passivates	The number of times that beans were passivated (entity and stateful)	3.5.5 and later	Per home	CountStatistic	Low
Num activates	beanModule.activates	The number of times that beans were activated (entity and stateful)	3.5.5 and later	Per home	CountStatistic	Low
Num loads	beanModule.loads	The number of times that bean data was loaded from persistent storage (entity)	3.5.5 and later	Per home	CountStatistic	Low
Num stores	beanModule.stores	The number of times that bean data was stored in persistent storage (entity)	3.5.5 and later	Per home	CountStatistic	Low
Num instantiates	beanModule.instantiates	The number of times that bean objects were instantiated	3.5.5 and later	Per home	CountStatistic	Low
Num destroys	beanModule.destroys	The number of times that bean objects were freed	3.5.5 and later	Per home	CountStatistic	Low
Ready beans	beanModule.readyCount	The number of concurrently ready beans (entity and session). This counter was called concurrent active in Versions 3.5.5+ and 4.0.	3.5.5 and later	Per home	RangeStatistic	High
Concurrent lives	beanModule.concurrentLives	The number of concurrently live beans	3.5.5 and later	Per home	RangeStatistic	High
Avg method RT (ms)	beanModule.avgMethodRt	The average response time in milliseconds on the bean methods (home, remote, local)	3.5.5 and later	Per home	TimeStatistic	High
Avg create time (ms)	beanModule.avgCreateTime	The average time in milliseconds that a bean create call takes including the time for the load if any	5.0	Per home	TimeStatistic	Max
Avg load time (ms)	beanModule.loadTime	The average time in milliseconds for loading the bean data from persistent storage (entity)	5.0	Per home	TimeStatistic	Medium
Avg store time (ms)	beanModule.storeTime	The average time in milliseconds for storing the bean data to persistent storage (entity)	5.0	Per home	TimeStatistic	Medium
Avg remove time (ms)	beanModule.avgRemoveTime	The average time in milliseconds that a bean entries call takes including the time at the database, if any	5.0	Per home	TimeStatistic	Max

Total method calls	beanModule.totalMethodCalls	The total number of method calls	3.5.5 and later	Per home	CountStatistic	High
Activation time (ms)	beanModule.activationTime	The average time in milliseconds that a beanActivate call takes including the time at the database, if any	5.0	Per home	TimeStatistic	Medium
Passivation time (ms)	beanModule.passivationTime	The average time in milliseconds that a beanPassivate call takes including the time at the database, if any	5.0	Per home	TimeStatistic	Medium
Active methods	beanModule.activeMethods	The number of concurrently active methods - the number of methods called at the same time.	3.5.5 and later	Per home	TimeStatistic	High
Gets from pool	beanModule.getsFromPool	The number of calls retrieving an object from the pool (entity and stateless)	3.5.5 and later	Per home and per object pool	CountStatistic	Low
Gets found	beanModule.getsFound	The number of times that a retrieve found an object available in the pool (entity and stateless)	3.5.5 and later	Per home and per object pool	CountStatistic	Low
Returns to pool	beanModule.returnsToPool	The number of calls returning an object to the pool (entity and stateless)	3.5.5 and later	Per home and per object pool	CountStatistic	Low
Returns discarded	beanModule.returnsDiscarded	The number of times that the returning object was discarded because the pool was full (entity and stateless)	3.5.5 and later	Per home and per object pool	CountStatistic	Low
Drains from pool	beanModule.drainsFromPool	The number of times that the daemon found the pool was idle and attempted to clean it (entity and stateless)	3.5.5 and later	Per home and per object pool	CountStatistic	Low
Avg drain size	beanModule.avgDrainSize	The average number of objects discarded in each drain (entity and stateless)	3.5.5 and later	Per home and per object pool	TimeStatistic	Medium
Pool size	beanModule.poolSize	The number of objects in the pool (entity and stateless)	3.5.5 and later	Per home and per object pool	RangeStatistic	High
Message count	beanModule.messageCount	The number of messages delivered to the bean onMessage method (message driven beans)	5.0	Per type	CountStatistic	Low

Message backout count	beanModule.messageBackoutCount	The number of messages that failed to be delivered to the bean onMessage method (message driven beans)	5.0	Per type	CountStatistic	Low
Server session wait time (ms)	beanModule.avgSrvSessionWaitTime	The average time to obtain a ServerSession from the pool (message drive bean)	5.0	Per type	TimeStatistic	Medium
Server session usage	beanModule.serverSessionUsage	The percentage of the server session pool in use (message driven)	5.0	Per type	RangeStatistic	High

Enterprise Java Bean method counters

Counter definitions:

Name	Key	Description	Version	Granularity	Type	Level
Method Calls	beanModule.methods.methodCalls	The number of calls to the bean methods (home, remote, local)	3.5.5 and later	Per method or per home	CountStatistic	Max
Method Response Time (ms)	beanModule.methods.methodRt	The average response time in milliseconds on the bean methods (home, remote, local)	3.5.5 and later	Per method or per home	TimeStatistic	Max
Concurrent Invocations	beanModule.method.methodLoad	The number of concurrent invocations to call a method	5.0	Per method or per home	RangeStatistic	Max

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**.

Counter definitions

Name	Key	Description	Version	Granularity	Type	Level
Num creates	connectionPoolModule.numCreates	The total number of connections created	3.5.5 and later	Per connection pool	CountStatistic	Low
Pool size	connectionPoolModule.poolSize	The size of the connection pool	3.5.5 and later	Per connection pool	BoundedRangeStatistic	High
Free pool size	connectionPoolModule.freePoolSize	The number of free connections in the pool	5.0	Per connection pool	BoundedRangeStatistic	High

Num allocates	connectionPoolModule.numAllocates	The total number of connections allocated	3.5.5 and later	Per connection pool	CountStatistic	Low
Num returns	connectionPoolModule.numReturns	The total number of connections returned	4.0 and later	Per connection pool	CountStatistic	Low
Concurrent waiters	connectionPoolModule.concurrentWaiters	The number of threads that are currently waiting for a connection	3.5.5 and later	Per connection pool	RangeStatistic	High
Faults	connectionPoolModule.faults	The total number of faults, such as timeouts, in the connection pool	3.5.5 and later	Per connection pool	CountStatistic	Low
Num closes	connectionPoolModule.numDestroys	The number of times freed bean objects	3.5.5 and later	Per connection pool	CountStatistic	Low
Avg wait time (ms)	connectionPoolModule.avgWaitTime	The average waiting time in milliseconds until a connection is granted	5.0	Per connection pool	TimeStatistic	Medium
Avg use time (ms)	connectionPoolModule.avgUseTime	The average time a connection is used (Difference between the time at which the connection is allocated and returned. This value includes the JDBC operation time.)	5.0	Per connection pool	TimeStatistic	Medium
Percent used	connectionPoolModule.percentUsed	The average percent of the pool that is in use	3.5.5 and later	Per connection pool	RangeStatistic	High
Percent maxed	connectionPoolModule.percentMaxed	The average percent of the time that all connections are in use	3.5.5 and later	Per connection pool	RangeStatistic	High
Prepared stmt cache discards	connectionPoolModule.prepStmtCacheDiscards	The total number of statements discarded by the least recently used (LRU) algorithm of the statement cache	4.0 and later	Per connection pool	CountStatistic	Low
Num managed connections	connectionPoolModule.numManagedConnections	The number of ManagedConnection objects in use for a particular connection pool (applies to V5.0 DataSource objects only)	5.0	Per connection factory	CountStatistic	Low

Num connection handles	connectionPoolModule.numConnectionHandles	The number of Connection objects in use for a particular connection pool (apply to 5.0 DataSource only)	5.0	Per connection factory	CountStatistic	Low
JDBC time	connectionPoolModule.jdbcOperationTimer	The amount of time in milliseconds spent running in the JDBC driver (includes time spent in the JDBC driver, network, and database)	5.0	Per data source	TimeStatistic	Medium

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
Num Managed Connections	j2cModule.numManagedConnections	The number of ManagedConnection objects in use	5.0	Per connection factory	CountStatistic	Low
Num Connection Handles	j2cModule.numConnectionHandles	The number of connections that are associated with ManagedConnections (physical connections) objects in this pool	5.0	Per connection factory	CountStatistic	Low
Num Connections Created	j2cModule.numManagedConnectionsCreated	The total number of managed connections created	5.0	Per connection factory	CountStatistic	Low
Num Connections Destroyed	j2cModule.numManagedConnectionsDestroyed	The total number of managed connections destroyed	5.0	Per connection factory	CountStatistic	Low
Num Connections Allocated	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	Low
Num Connections Freed	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	Low
Num Faults	j2cModule.faults	The number of faults, such as timeouts, in the connection pool	5.0	Per connection factory	CountStatistic	Low
Free Pool Size	j2cModule.freePoolSize	The number of free connections in the pool	5.0	Per connection factory	BoundedRangeStatistic	High
Pool Size	j2cModule.poolSize	Average number of managed connections in the pool.	5.0	Per connection factory	BoundedRangeStatistic	High

Concurrent Waiters	j2cModule.concurrentWaiters	Average number of threads concurrently waiting for a connection	5.0	Per connection factory	RangeStatistic	High
Percent Used	j2cModule.percentUsed	Average percent of the pool that is in use	5.0	Per connection factory	RangeStatistic	High
Percent Maxed	j2cModule.percentMaxed	Average percent of the time that all connections are in use	5.0	Per connection factory	RangeStatistic	High
Avg Wait Time	j2cModule.avgWait	Average waiting time in milliseconds until a connection is granted	5.0	Per connection factory	TimeStatistic	Medium
Avg Use Time	j2cModule.useTime	Average time in milliseconds that connections are in use	5.0	Per connection factory	TimeStatistic	Medium

Servlet session counters

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

Counter definitions

Name	Key	Description	Version	Granularity	Type	Level
Created sessions	servletSessionsModule.createdSessions	The number of sessions that were created	3.5.5 and later	Per Web application	CountStatistic	Low
Invalidated sessions	servletSessionsModule.invalidatedSessions	The number of sessions that were invalidated	3.5.5 and later	Per Web application	CountStatistic	Low
Session life time (ms)	servletSessionsModule.sessionLifeTime	The average session life time in milliseconds (time invalidated - time created)	3.5.5 and later	Per Web application	TimeStatistic	Medium
Active sessions	servletSessionsModule.activeSessions	The number of concurrently active sessions. A session is active if the WebSphere Application Server is currently processing a request that uses that session.	3.5.5 and later	Per Web application	RangeStatistic	High
Live sessions	servletSessionsModule.liveSessions	The number of sessions that are currently cached in memory	5.0 and later	Per Web application	RangeStatistic	High
No room for new sessions	servletSessionsModule.noRoomForNewSession	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.	5.0	Per Web application	CountStatistic	Low
Cache discards	servletSessionsModule.cacheDiscards	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.	5.0	Per Web application	CountStatistic	Low

External read time	servletSessionsModule.externalReadTime	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.	5.0	Per Web application	TimeStatistic	Medium
External read size	servletSessionsModule.externalReadSize	Size of the session data read from persistent store. Applicable only for (serialized) persistent sessions; similar to external Read Time.	5.0	Per Web application	TimeStatistic	Medium
External write time	servletSessionsModule.externalWriteTime	The time (milliseconds) taken to write the session data from the persistent store. Applicable only for (serialized) persistent sessions. Similar to external Read Time.	5.0	Per Web application	TimeStatistic	Medium
External write size	servletSessionsModule.externalWriteSize	The size of the session data written to persistent store. Applicable only for (serialized) persistent sessions. Similar to external Read Time.	5.0	Per Web application	TimeStatistic	Medium
Affinity breaks	servletSessionsModule.affinityBreaks	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.	5.0	Per Web application	CountStatistic	Low
Session object size	servletSessionsModule.serializableSessObjSize	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.	5.0	Per Web application	TimeStatistic	Max
Time since last activated	servletSessionsModule.timeSinceLastActivated	The time difference in milliseconds between previous and current access time stamps. Does not include session time out.	5.0	Per Web application	TimeStatistic	Medium

Invalidated via timeout	servletSessionsModule.invalidatedViaTimeout	The number of sessions that are invalidated by timeout.	5.0	Per Web application	CountStatistic	Low
Activate non-exist sessions	servletSessionsModule.activateNonExistSessions	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.	5.0	Per Web application	CountStatistic	Low

Transaction counters

Counter definitions

Name	Key	Description	Version	Granularity	Type	Level
Global trans begun	transactionModule.globalTransBegun	The total number of global transactions started on the server	4.0 and later	Per transaction manager or server	CountStatistic	Low
Global trans involved	transactionModule.globalTransInvolved	The total number of global transactions involved on the server (for example, begun and imported)	4.0 and later	Per transaction manager or server	CountStatistic	Low
Local trans begun	transactionModule.localTransBegun	The total number of local transactions started on the server	4.0 and later	Per transaction manager or server	CountStatistic	Low
Active global trans	transactionModule.activeGlobalTrans	The number of concurrently active global transactions	3.5.5 and later	Per transaction manager or server	CountStatistic	Low
Active local trans	transactionModule.activeLocalTrans	The number of concurrently active local transactions	4.0 and later	Per transaction manager or server	CountStatistic	Low
Global tran duration	transactionModule.globalTranDuration	The average duration of global transactions	3.5.5 and later	Per transaction manager or server	TimeStatistic	Medium
Local tran duration	transactionModule.localTranDuration	The average duration of local transactions	4.0 and later	Per transaction manager or server	TimeStatistic	Medium
Global before completion duration	transactionModule.globalBeforeCompletionDuration	The average duration of before_completion for global transactions	4.0 and later	per transaction manager or server	TimeStatistic	Medium
Global commit duration	transactionModule.globalCommitDuration	The average duration of commit for global transactions	4.0 and later	Per transaction manager or server	TimeStatistic	Medium
Global prepare duration	transactionModule.globalPrepareDuration	The average duration of prepare for global transactions	4.0 and later	Per transaction manager or server	TimeStatistic	Medium
Local before completion duration	transactionModule.localBeforeCompletionDuration	The average duration of before_completion for local transactions	4.0 and later	Per transaction manager or server	TimeStatistic	Medium
Local commit duration	transactionModule.localCommitDuration	The average duration of commit for local transactions	4.0 and later	Per transaction manager or server	TimeStatistic	Medium
Global trans committed	transactionModule.globalTransCommitted	The total number of global transactions committed	3.5.5 and later	Per transaction manager or server	CountStatistic	Low
Global trans rolled back	transactionModule.globalTransRolledBack	The total number of global transactions rolled back	3.5.5 and later	Per transaction manager or server	CountStatistic	Low
Num optimizations	transactionModule.numOptimization	The number of global transactions converted to single phase for optimization	4.0 and later	Per transaction manager or server	CountStatistic	Low

Local trans committed	transactionModule.localTransCommitted	The number of local transactions committed	4.0 and later	Per transaction manager or server	CountStatistic	Low
Local trans rolled back	transactionModule.localTransRolledBack	The number of local transactions rolled back	4.0 and later	Per transaction manager or server	CountStatistic	Low
Global trans timeout	transactionModule.globalTransTimeout	The number of global transactions timed out	4.0 and later	Per transaction manager or server	CountStatistic	Low
Local trans timeout	transactionModule.localTransTimeout	The number of local transactions timed out	4.0 and later	Per transaction manager or server	CountStatistic	Low

Web application counters

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

Counter definitions

Name	Key	Description	Version	Granularity	Type	Level
Num loaded servlets	webAppModule.numLoadedServlets	The number of loaded servlets	3.5.5 and later	Per Web application	CountStatistic	Low
Num reloads	webAppModule.numReloads	The number of reloaded servlets	3.5.5 and later	Per Web application	CountStatistic	Low
Total requests	webAppModule.servlets.totalRequests	Total number of requests that a servlet processed	3.5.5 and later	Per servlet	CountStatistic	Low
Concurrent requests	webAppModule.servlets.concurrentRequests	The number of requests that are concurrently processed	3.5.5 and later	Per servlet	RangeStatistic	High
Average response time (ms)	webAppModule.servlets.responseTime	The response time, in milliseconds, of a servlet request	3.5.5 and later	Per servlet	TimeStatistic	Medium
Num errors	webAppModule.servlets.numErrors	Total number of errors in a servlet or JavaServer Page (JSP)	3.5.5 and later	Per servlet	CountStatistic	Low

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 Configuring the dynamic cache service to improve performance.

Use the DynaCache MBean to access the related data and display it under Dynamic Cache in TPV.

Counter definitions

Name	Key	Description	Version	Granularity	Type	Level
Max in memory cache size	cacheModule.maxInMemoryCacheSize	The maximum number of in-memory cache entries	5.0	Per server	CountStatistic	Low
In memory cache size	cacheModule.inMemoryCacheSize	The current number of in-memory cache entries	5.0	Per server	CountStatistic	Low
Timeouts	cacheModule.totalTimeoutInvalidations	The aggregate of template timeouts and disk timeouts	5.0	Per server	CountStatistic	Low

Hits in memory	cacheModule.template.hitsInMemory	The requests for this cacheable object served from memory	5.0	Per template	CountStatistic	Low
Hits on disk	cacheModule.template.hitsOnDisk	The requests for this cacheable object served from disk	5.0	Per template	CountStatistic	Low
Explicit invalidations	cacheModule.template.explicitInvalidations	The total of explicit invalidations issued for this template	5.0	Per template	CountStatistic	Low
LRU invalidations	cacheModule.template.lruInvalidations	Cache entries evicted from memory by a least recently used algorithm. These entries are passivated to disk if disk overflow is enabled.	5.0	Per template	CountStatistic	Low
Timeouts	cacheModule.template.timeoutInvalidations	The cache entries evicted from memory or disk, or both, because their timeout has expired	5.0	Per template	CountStatistic	Low
Entries	cacheModule.template.entries	The current number of cache entries created from this template. This refers to the per-template equivalent of totalCacheSize.	5.0	Per template	CountStatistic	Low
Misses	cacheModule.template.misses	The requests for this cacheable object that were not found in the cache	5.0	Per template	CountStatistic	Low
Client requests	cacheModule.template.requestsFromClient	The requests for this cacheable object generated by applications running on the application server	5.0	Per template	CountStatistic	Low
Distributed requests	cacheModule.template.requestsFromJVM	The requests for this cacheable object generated by cooperating caches in this cluster	5.0	Per template	CountStatistic	Low
Explicit invalidations (memory)	cacheModule.template.explicitInvalidationsFromMemory	The explicit invalidations resulting in an entry being removed from memory	5.0	Per template	CountStatistic	Low
Explicit invalidations (disk)	cacheModule.template.explicitInvalidationsFromDisk	The explicit invalidations resulting in an entry being removed from disk	5.0	Per template	CountStatistic	Low
Explicit invalidations (no op)	cacheModule.template.explicitInvalidationsNoOp	The explicit invalidations received for this template where no corresponding entry exists	5.0	Per template	CountStatistic	Low
Local explicit invalidations	cacheModule.template.explicitInvalidationsLocal	The explicit invalidations generated locally, either programmatically or by a cache policy	5.0	Per template	CountStatistic	Low
Remote explicit invalidations	cacheModule.template.explicitInvalidationsRemote	The explicit invalidations received from a cooperating JVM in this cluster	5.0	Per template	CountStatistic	Low

Remote creations	cacheModule.template.remoteCreations	The entries received from cooperating dynamic caches	5.0	Per template	CountStatistic	Low

Web services counters

Counters for this category contain information for the Web services.

Counter definitions

Name	Key	Description	Version	Granularity	Type	Level
Number of Web services	webServicesModule.numLoadedServices	The number of loaded Web services	5.02 and above	Per service	CountStatistic	Low
Number of requests received	webServicesModule.services.numberReceived	The number of requests the service received	5.02 and above	Per Web service	CountStatistic	Low
Number of requests dispatched	webServicesModule.services.numberDispatched	The number of requests the service dispatched or delivered	5.02 and above	Per Web service	CountStatistic	Low
Number of requests processed successfully	webServicesModule.services.numberSuccessfull	The number of requests the service successfully processed	5.02 and above	Per Web service	TimeStatistic	Low
Average response time	webServicesModule.services.responseTime	The average response time, in milliseconds, for a successful request	5.02 and above	Per Web service	TimeStatistic	High
Average request response time	webServicesModule.services.requestResponseTime	The average response time, in milliseconds, to prepare a request for dispatch	5.02 and above	Per Web service	TimeStatistic	Medium
Average dispatch response time	webServicesModule.services.dispatchResponseTime	The average response time, in milliseconds, to dispatch a request	5.02 and above	Per Web service	TimeStatistic	Medium
Average reply response time	webServicesModule.services.replyResponseTime	The average response time, in milliseconds, to prepare a reply after dispatch	5.02 and above	Per Web service	TimeStatistic	Medium
Average payload size	webServicesModule.services.size	The average payload size in bytes of a received request or reply	5.02 and above	Per Web service	TimeStatistic	Medium
Average request payload size	webServicesModule.services.requestSize	The average payload size in bytes of a request	5.02 and above	Per Web service	TimeStatistic	Medium
Average reply payload size	webServicesModule.services.replySize	The average payload size in bytes of a reply	5.02 and above	Per Web service	TimeStatistic	Medium

Performance data classification

Performance Monitoring Infrastructure provides server-side data collection and client-side API to retrieve performance data. Performance data has two components: static and dynamic.

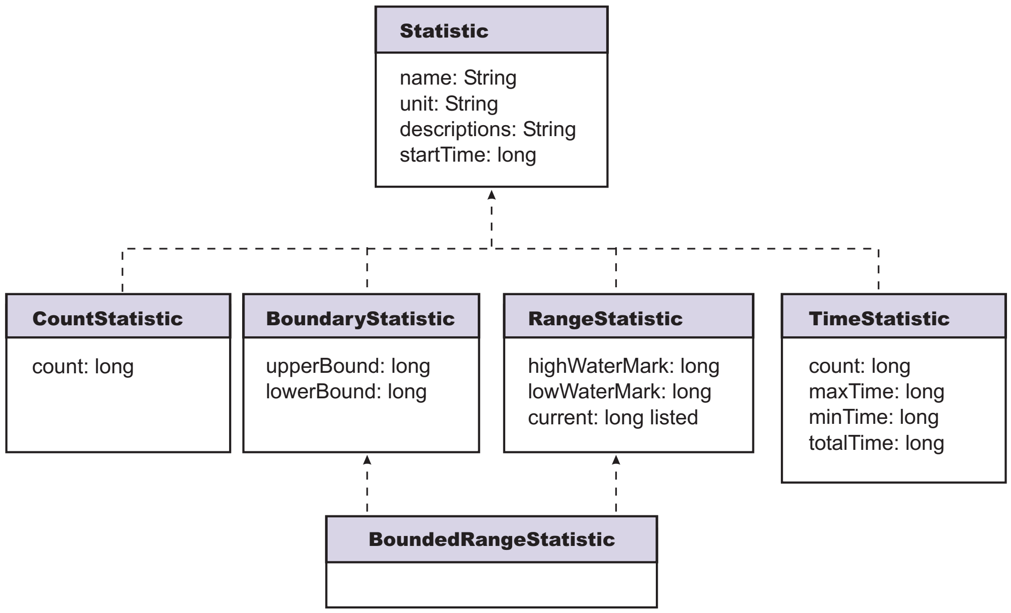
The static component consists of a name, ID and other descriptive attributes to identify the data. The dynamic component contains information that changes over time, such as the current value of a counter and the time stamp associated with that value.

The PMI data can be one of the following statistical types defined in the JSR-077 specification:
- CountStatistic
- BoundaryStatistic
- RangeStatistic
- TimeStatistic
- BoundedRangeStatistic

RangeStatistic data contains current value, as well as lowWaterMark and highWaterMark.

In general, 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 Profiler Interface (JVMPI) counters, SerializableSessObjSize, and data tracked for each individual method (method level data) require a *maximum* monitoring level.



In previous versions, PMI data was classified with the following types:
- **Numeric**: Maps to CountStatistic in the JSR-077 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 JSR-077 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 API. Statistical data types are supported through both the PMI API and Java Management Extension (JMX) API.

The TimeStatistic type keeps tracking many counter samples and then returns the total, count and average of the samples. An example of this is an average method response time. Given the nature of this statistic type, it is also used to track non-time related counters, like average read and write size. You can always call getUnit method on the data configuration information to learn the unit for the counter.

In order to reduce the monitoring overhead, numeric and stat data are not synchronized. Since these data track the total and average, the extra accuracy is generally not worth the performance cost. Load data is very sensitive, therefore, load counters are always synchronized. In addition, when the monitoring level of a module is set to *max*, all numeric data are also synchronized to guarantee accurate values.

Enabling performance monitoring services in the application server through the administrative console

To monitor performance data through the performance monitoring infrastructure (PMI) interfaces, you must first enable PMI services through the administrative console.

1. Open the administrative console.
2. Click **Servers** > **Application Servers** in the console navigation tree.
3. Click *server*.
4. Click the **Configuration** tab. When in the Configuration tab, settings will apply once the server is restarted. When in the Runtime Tab, settings will apply immediately. Note that enablement of Performance Monitoring Service can only be done in the Configuration tab.
5. Click **Performance Monitoring Service**.
6. Select the checkbox **Startup**.
7. (Optional) Select the PMI modules and levels to set the **initial specification level** field.
8. Click **Apply** or **OK**.
9. Click **Save**.
10. Restart the application server. The changes you make will not take affect until you restart the application server.

When running in WebSphere Application Server Network Deployment, refer to "Enabling performance monitoring services in the Node Agent through the administrative console" on page 22.

Note: If you are running your monitoring applications with security enabled, refer to "Running your monitoring applications with security enabled" on page 71.

Performance monitoring service settings

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

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

Startup

Specifies whether the application server attempts to start the specified service. If an application server is started when the performance monitoring service is disabled, you will have to restart the server in order to enable it.

Initial specification level

Specifies a Performance Monitoring Infrastructure (PMI) string that stores PMI specification levels, for example module levels, for all components in the server.

Set the PMI specification levels by selecting the *none*, *standard* or *custom* checkbox. If you choose *none*, all PMI modules are set to the *none* level. Choosing *standard*, sets all PMI modules to *high* and enables all PMI data excluding the method level data and JVMPI data. Choosing custom, gives you the option to change the level for each individual PMI module. You can set the level to N, L, M, H or X (none, low, medium, high and maximum). **Note** that you should not change the module names.

Enabling performance monitoring services in the Node Agent through the administrative console

To monitor performance data through the performance monitoring infrastructure (PMI) interfaces, you must first enable PMI services through the administrative console.

1. Open the administrative console.
2. Click **System Administration** > **Node Agents** in the console navigation tree.
3. Click *nodeagent*.
4. Click **Performance Monitoring Service**.
5. Select the **Startup** check box .
6. (Optional) Select the PMI modules and levels to set the **initial specification level** field.
7. Click **Apply** or **OK**.
8. Click **Save**.
9. Restart the Node Agent. The changes take affect after you restart the Node Agent.

When in the Configuration tab, settings will apply once the server is restarted. When in the Runtime Tab, settings will apply immediately. Note that enablement of Performance Monitoring Service can only be done in the Configuration tab.

Note: If you are running your monitoring applications with security enabled, refer to Running your monitoring applications with security enabled.

Enabling performance monitoring services using the command line

You can use the command line to enable performance monitoring services.

1. Enable PMI services as described in“Enabling performance monitoring services in the application server through the administrative console” on page 21. Make sure to restart the application server.

2. Run the **wsadmin** command as described in Launching scripting clients. 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**:

   ```
   /** Set instrumentation level using String format
   *  This should be used by scripting for an easy String processing
   */ The level STR is a list of moduleName=Level connected by ":".
   */
   public void setInstrumentationLevel(String levelStr, Boolean recursive);

   /** Get instrumentation level in String for all the top level modules
   *  This should be used by scripting for an easy String processing
   */
   public String getInstrumentationLevelString();

   /** Return the PMI data in String
   *
   */
   public String getStatsString(ObjectName on, Boolean recursive);

   /** Return the PMI data in String
   *  Used for PMI modules/submodules without direct MBean mappings.
   */
   public String getStatsString(ObjectName on, String submoduleName, Boolean recursive);

   /**
   * Return the submodule names if any for the MBean
   */
   public String listStatMemberNames(ObjectName on);
   ```

   If an MBean is a StatisticProvider and if you pass its ObjectName to getStatsString, you will 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
   - TransactionService
   - WebModule
   - Servlet
   - WLMAppServer
   - WebServicesService
   - WSGW

The following are sample commands in **wsadmin** you can use to obtain PMI data:

Obtain the Perf MBean ObjectName

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

Invoke getInstrumentationLevelString operation

- use invoke since it has no parameter

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

This command returns the following:

```
beanModule=H:cacheModule=H:connectionPoolModule=H:j2cModule=H:jvmRuntimeModule=H
:orbPerfModule=H:servletSessionsModule=H:systemModule=H:threadPoolModule=H
:trans actionModule=H:webAppModule=H
```

Note that 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

- set 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 since it has parameter

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

This command does not return anything.

Note that 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 - e.g., 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 a MBean can be found by invoking listStatMemberNames. For example, beanModule is a logic module aggregating PMI data over all EJBs 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 - e.g., 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 EJBs within the BeanModule hierarchy since the recursive flag is set to true.

Note that 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
```

Monitoring and analyzing performance data

WebSphere Application Server performance data, once collected, can be monitored and analyzed with a variety of tools.

1. Monitor performance with Tivoli Performance Viewer as described in "Monitoring performance with Tivoli Performance Viewer (formerly Resource Analyzer)." This tool is included with WebSphere Application Server.
2. Write your own applications to monitor performance data as described in "Developing your own monitoring applications" on page 37.
3. Use third-party monitoring tools as explained in "Third-party performance monitoring and management solutions" on page 72.
4. Use RMF as discussed in "RMF Workload Activity reports and RMF Monitor III" on page 72.
5. Refer to "WLM Delay Monitoring" on page 74.

Monitoring performance with Tivoli Performance Viewer (formerly Resource Analyzer)

The Resource Analyzer has been renamed *Tivoli Performance Viewer*.

Tivoli Performance Viewer (which is shipped with WebSphere) is a Graphical User Interface (GUI) performance monitor for WebSphere Application Server. Tivoli Performance Viewer can connect to a local or to a remote host. Connecting to a remote host will minimize performance impact to the application server environment.

Monitor and analyze the data with Tivoli Performance Viewer with these tasks:

1. "Starting the Tivoli Performance Viewer" on page 31.
2. "Setting performance monitoring levels" on page 32.
3. "Viewing summary reports" on page 33.
4. Optional: "Storing data to a log file" on page 36.
5. Optional: "Replaying a performance data log file" on page 37.
6. Optional: "Viewing and modifying performance chart data" on page 34.
7. Optional: "Scaling the performance data chart display" on page 34.
8. Optional: "Refreshing data" on page 35.
9. Optional: "Clearing values from tables and charts" on page 35.
10. Optional: "Resetting counters to zero" on page 37.

Tivoli Performance Viewer features

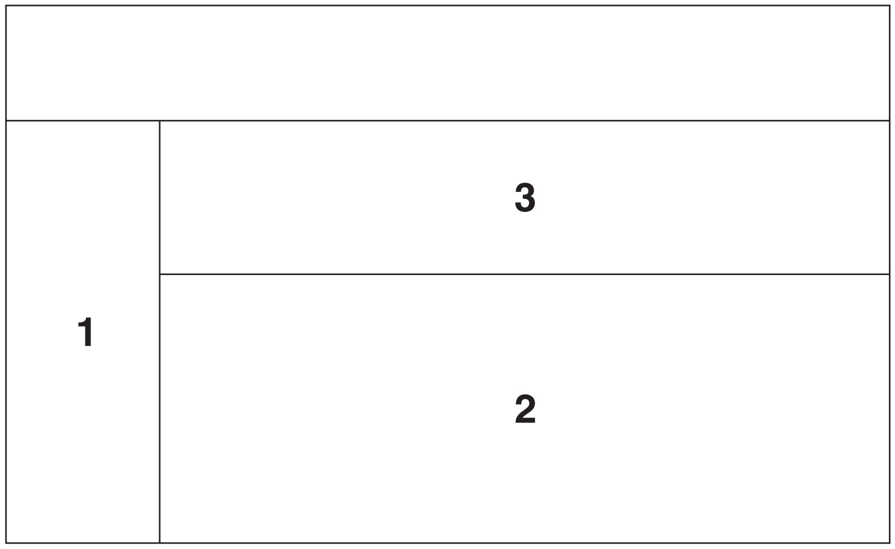
Tivoli Performance Viewer is a Java client which retrieves the Performance Monitoring Infrastructure (PMI) data from an application server and displays it in a variety of formats.

You can do the following tasks with the Tivoli Performance Viewer:

- View data in real time
- Record current data in a log, and replay the log later
- View data in chart form, allowing visual comparison of multiple counters. Each counter can be scaled independently to enable meaningful graphs.
- View data in tabular form
- Compare data for single resources to aggregate data across a node

To minimize the performance impact, Tivoli Performance Viewer polls the server with the PMI data at an interval set by the user. All data manipulations are done in the Tivoli Performance Viewer client, which can be run on a separate machine, further reducing the impact.

The Tivoli Performance Viewer graphical user interface includes the following:
- Resource selection panel
- Data monitoring panel
- Menu bar
- Toolbar icons
- Node icons
- Status bar



1 - Resource Selection Panel
2 - Counter Selection panel
3 - Viewing Counter (chart and table views)

Layout of the console

The performance viewer main window consists of two panels: the Resource Selection panel and the Data Monitoring panel. The Resource Selection panel, located on the left, provides a view of resources for which performance data can be displayed. The Data Monitoring panel, located on the right, displays numeric and statistical data for the resources that are highlighted (selected) in the Resource Selection panel.

You can adjust the width of the Resource Selection and Data Monitoring panels by dragging the split bar left or right. You can rearrange the order of the table columns in the Data Monitoring panel by dragging the column heading left or right. You can also adjust the width of the columns by dragging the edge of the column left or right.

Resource selection panel

The Resource Selection panel provides a hierarchical (tree) view of resources and the types of performance data available for those resources. Use this panel to select which resources to monitor and to start and stop data retrieval for those resources.

The Resource Selection panel displays resources and associated resource categories in an indented tree outline. Clicking the plus (+) and minus (-) symbols expands and collapses the tree to reveal the categories for the various resource instances. The resource tree can also be navigated by using the up and down arrow keys to cycle through the branches and by using the left and right arrow keys to expand and collapse the tree of resources. Resource instances can be expanded to reveal the instances they contain, if applicable. For example, when a EJB JAR instance is expanded, the enterprise bean instances in the EJB JAR are revealed. The Data Monitoring panel automatically displays the appropriate selection of counters for any objects highlighted in the Resource Selection panel.

The first level of the hierarchy includes all nodes (machines) in the administrative domain, followed by all application servers on the node. Below each application server, all resource categories are listed. If the enterprise beans category is expanded, all EJB JAR instances in the server are displayed. Next, all enterprise bean instances appear below the EJB JAR in the hierarchy. Then, a methods resource is associated with each bean. Clicking an individual bean or EJB JAR instance causes its corresponding counters to be displayed in the Data Monitoring panel. For enterprise beans, the counters displayed depend on whether the bean is an entity bean or a session bean. For EJB JARs, the counters are aggregate counters for all enterprise beans in the EJB JARs. See the information center article Performance data organization for more information.

Data monitoring panel

The Data Monitoring panel enables the selection of multiple counters and displays the resulting performance data for the currently selected resource. It contains two panels: the Viewing Counter panel above and the Counter Selection panel below.

Counter selection panel

The Counter Selection panel shows the counters available for the resource performance category selection.

Two factors determine the list of available counters in the Counter Selection panel:
- Only counters associated with the resource that is selected in the Resource Selection panel are displayed.
- Only counters having impact cost ratings within or below the instrumentation or monitoring level that is set for that resource in the administrative domain are displayed.

The first three counters shown for each resource performance category are selected by default. All counters can be selected or deselected, and the resulting output, shown in the top panel, automatically reflects the selection.

The columns in the Counter Selection panel provide the following information for each counter:
- **Name**. The names of the counters that are available for selection with this resource.
- **Description**. A brief description of the function of each counter.
- **Value**. The value for the counter, displayed according to the display mode in effect. Values are actual values (not scaled values used for the chart, if applicable).
- **Select**. A check box that indicates whether a counter is to be reflected in the chart. To hide data, clear the check box. The column representing that counter is then removed from the View Data window, and the graphic display for that counter is removed from the View Chart window.
- **Scale**. A value indicating whether data has been scaled (amplified or diminished) from its actual value to fit on the chart. This value is reflected only in the View Chart window.

  The value for the **Scale** column can be set manually by editing the value of the **Scale** field. See Scaling the chart display manually for information on manually setting the scale.

Viewing Counter panel

When a counter on the list in the Counter Selection panel is selected, the statistics gathered from that counter are displayed in the Viewing Counter panel at the top of the Data Monitoring panel.

The View Data window shows the counter's output in table format; the View Chart window displays a graph with time represented on the *x-axis* and the performance value represented on the *y-axis*. One or more performance counters can be simultaneously graphed on a single chart. The chart plots data from *n* data points, where *n* is the current table size (number of rows).

Display of multiple resources and aggregate data

When a single resource is selected in the Resource Selection panel, the Data Monitoring panel displays a choice of a table view or a chart view. If multiple resources are selected, the Data Monitoring panel displays a single data sheet for viewing summary information for the selected resources. The data sheet displays the tables for all objects of similar type for the selected resources. For example, if three servlet instances are selected, the data sheet displays a table of counter values for all the servlets. By default, the display buffer size is set to 40 rows, corresponding to the values of the last 40 data points retrieved.

The performance viewer provides aggregate data at the module level. If aggregate data is available for a group, it is displayed in the Data Monitoring panel. For example, for each enterprise bean home interface, counters track the number of active enterprise beans of that home. Each EJB JAR has an aggregate value that is the sum of all the enterprise beans in that EJB JAR. The enterprise beans resource category (module) within the application server has an aggregate value that is the sum of all enterprise beans in all EJB JARs.

Menu bar

The menu bar contains the following options:
- **File** menu. Used to change to current mode (from logging mode), to open an existing log file, and to exit from the performance viewer. The **File** menu contains the following items:

  - **Refresh**. Queries the administrative server for any newly started resources since data retrieval began or for additional counters to report. This operation is also recursive over all components subordinate to the selected resources. Tivoli Performance Viewer refreshes data every 10 seconds. When changing the refresh rate, you must use an integer greater than or equal to 1.
  - **Current Activity**. Resumes the display of real-time data in tables and charts. This menu option is used to stop viewing data from a log file and return to viewing real-time data.
  - **Log**. Displays a dialog box for specifying the name and location of an existing log file to be replayed.
  - **Exit**. Closes the performance viewer. If you made changes to the instrumentation levels of any resources during the session, a dialog box opens to ask whether you want to save the changed settings before closing the tool.
- **Logging** menu. Provides **On** and **Off** options that are used to start and stop recording data in a log file. If you start a new log file and specify the same file name, the file is overwritten.
- **Setting** menu. Used to start and stop the reporting of data, and to clear and refresh data. The **Setting** menu contains the following items:
  - **Clear Buffer**. Deletes the values currently displayed in tables and charts. For example, after stopping a counter, you can use this operation to remove the remaining data from a table.
  - **Reset to Zero**. Resets cumulative counters of the selected performance group back to zero.
  - **View Data As**. Specifies how counter values are displayed. You can choose whether to display absolute values, changes in values, or rates of change. How data is displayed differs slightly depending on where you are viewing data. The choices follow:
    - **Raw Value**. Displays the absolute value. If the counter represents load data, such as the average number of connections in a database pool, then the Tivoli Performance Viewer 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.
  - **Log Replay**. Includes **Rewind Stop Play Fast Forward**.

  Note that right-clicking a resource in the Resource Selection panel displays a menu that provides the following options: **Refresh**, **Clear Buffer**, and **Reset to Zero**.
- **Help** menu. Provides information for users.

Toolbar icons

Toolbar icons provide shortcuts to frequently used commands. The toolbar includes the following icons:
- **Refresh**. Updates data and structures for the selected resources. That is, it polls the administrative server to retrieve new information about additional counters to display or new servers recently added to the domain.
- **Clear Buffer**. Deletes the values currently displayed in all tables and charts.
- **Reset to Zero**. Resets the counters.

Node icons

In the Resource Selection panel, the color of the node icon indicates the current state and availability of the application server in the domain.
- Green--The resource is running and available.
- Red--The resource is stopped.

Status bar

The status bar across the bottom of the performance viewer window dynamically displays the current state of the reporting values. The following state information is reported in the status bar:
- The current setting for the refresh rate
- The buffer size in use in the current Viewing Counter panel
- The display mode in use in the current Viewing Counter panel
- The current state of the logging setting

Starting the Tivoli Performance Viewer

You can also start the Tivoli Performance Viewer with security enabled. To do this see "Running your monitoring applications with security enabled" on page 71.

1. Download the utility package that contains the Tivoli Performance Viewer.
   a. This file can be found in (assuming recommended install directory): /usr/lpp/zWebSphere/V5R0M0/bin/setup.exe
   b. Download it as a binary file and execute it on your Windows machine. Only the Windows platform is supported at this time.
2. Start the Tivoli Performance Viewer. This can be done in two ways:
   a. Start performance monitoring from the command line. Go to the `product_installation_directory/bin` directory and run the `tperfviewer` script.

      You can specify the host and port in Windows NT, 2000, and XP environments as:

      ```
      tperfviewer.bat host_name port_number connector_type
      ```

      or

      On the AIX and other UNIX platforms, use

      ```
      tperfviewer.sh host_name port_number connector_type
      ```

      for example:

      ```
      tperfviewer.bat localhost 8879 SOAP
      ```

      Connector_type can be either SOAP or RMI. The port numbers for SOAP/RMI connector can be configured in the Administrative Console under **Servers** > **Application Servers** > **server_name** > **End Points**.

      If you are connecting to WebSphere Application Server, use the application server host and connector port. If additional servers have been created, then use the appropriate server port for which data is required. Tivoli Performance Viewer will only display data from one server at a time when connecting to WebSphere Application Server.

      If you are connecting to WebSphere Application Server Network Deployment, use the deployment manager host and connector port. Tivoli Performance Viewer will display data from all the servers in the cell. Tivoli Performance Viewer cannot connect to an individual server in WebSphere Application Server Network Deployment.

      8879 is the default SOAP connector port for WebSphere Application Server Network Deployment.

      8880 is the default SOAP connector port for WebSphere Application Server.

9809 is the default RMI connector port for WebSphere Application Server Network Deployment.

2809 is the default RMI connector port for WebSphere Application Server.

On iSeries, you can connect the Tivoli Performance Viewer to an iSeries instance from either a Windows, an AIX, or a UNIX client as described above. To discover the RMI or SOAP port for the iSeries instance, start Qshell and enter the following command:

```
product_installation_directory/bin/dspwasinst -instance myInstance
```

where
- *product_installation_directory* is your iSeries install directory
- *myInstance* is the instance used when you created iSeries instance.

b. Click **Start** > **Programs** > **IBM WebSphere** > **Application Server v.50** > **Tivoli Performance Viewer**.

Tivoli Performance Viewer detects which package of WebSphere Application Server you are using and connects using the default SOAP connector port. If the connection fails, a dialog is displayed to provide new connection parameters.

You can connect to a remote host or a different port number, by using the command line to start the performance viewer.

3. Adjust the data collection settings. Refer to the instructions in "Setting performance monitoring levels."

Setting performance monitoring levels

The monitoring settings determine which counters are enabled. Changes made to the settings from Tivoli Performance Viewer affect all applications that use the Performance Monitoring Infrastructure (PMI) data.

To view monitoring settings:

1. Choose the **Data Collection** icon on the Resource Selection panel. This selection provides two options on the Counter Selection panel. Choose the **Current Activity** option to view and change monitoring settings. Alternatively, use **File**> **Current Activity** to view the monitoring settings.
2. Set monitoring levels by choosing one of the following options:
   - **None**: Provides no data collection
   - **Standard**: Enables data collection for all modules with monitoring level set to high
   - **Custom**: Allows customized settings for each module

   These options apply to an entire application server.
3. (Optional) Fine tune the monitoring level settings.
   a. Click **Specify**. This sets the monitoring level to custom.
   b. Select a monitoring level. For each resource, choose a monitoring level of **None**, **Low**, **Medium**, **High** or **Maximum**. The dial icon will change to represent this level. **Note:** The instrumentation level is set recursively to all elements below the selected resource. You can override this by setting the levels for children AFTER setting their parents.
4. Click **OK**.
5. Click **Apply**.

If the instrumentation level excludes a counter, that counter does not appear in the tables and charts of the performance viewer. For example, when the instrumentation level is set to low, the thread pool size is not displayed because that counter requires a level of high.

Note that monitoring levels can also be set through the administrative console. See "Enabling performance monitoring services in the application server through the administrative console" on page 21 for more information.

Viewing 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. See "Setting performance monitoring levels" on page 32.

The standard monitoring level enables all reports except the report on Enterprise JavaBeans (EJB) methods. To enable an EJB methods report, use the custom monitoring setting and set the monitoring level to `Max` for the EJB module.

Tivoli Performance Viewer provides the following summary reports for each application server:

Enterprise beans

Enterprise beans show the total number of method calls, average response time, and multiplication of total method calls by average response time for all the enterprise beans in a table. Enterprise beans provide a sorting feature to help you find which enterprise bean is the slowest or fastest and which enterprise bean is called most frequently.

EJB Methods

EJB Methods show the total number of method calls, average response time, and multiplication of total method calls by average response time for the individual EJB methods in a table. EJB Methods provide a sorting feature to help you find which EJB method is the slowest or fastest and which EJB method is called most frequently.

Servlets

Servlets show the total number of requests, average response time, and multiplication of total requests by average response time for all the servlets in a table. Servlets provide a sorting feature to help you find which servlet is the slowest or fastest and which servlet is called most frequently.

Web Container Pool

Web Container Pool shows charts of pool size, active threads, average response time, and throughput in the Web container thread pool

Object Request Broker (ORB) Thread Pool

ORB Thread Pool shows charts of pool size, active threads, average response time, and throughput in the ORB thread pool.

Connection Pool

Connection Pool shows a chart of pool size and pool in use for each data source.

1. Click the application server icon in the navigation tree.
2. Click the appropriate column header to sort the columns in the report.

Changing the refresh rate of data retrieval

By default, the Tivoli Performance Viewer retrieves data every 10 seconds.

To change the rate at which data is retrieved:

1. Click **Setting** > **Set Refresh Rate**.
2. Type a positive integer representing the number of seconds in the **Set Refresh Rate** dialog box.
3. Click **OK**.

Changing the display buffer size

To change the size of the buffer and the number of rows displayed:

1. Click **Setting** > **Set Buffer Size**.
2. Type the number of rows to display in the **Set Buffer Size** dialog box.
3. Click **OK**.

Viewing and modifying performance chart data

The **View Chart** tab displays a graph with time as the *x-axis* and the performance value as the *y-axis*.

1. Click a resource in the Resource Selection panel. The Resource Selection panel, located on the left side, provides a hierarchical (tree) view of resources and the types of performance data available for those resources. Use this panel to select which resources to monitor and to start and stop data retrieval for those resources. See Tivoli Performance Viewer features for information on the Resource Selection panel.
2. Click the **View Chart** tab in the Data Monitoring panel. The Data Monitoring panel, located on the right side, enables the selection of multiple counters and displays the resulting performance data for the currently selected resource. It contains two panels: the Viewing Counter panel above and the Counter Selection panel below. If necessary, you can set the scaling factors by typing directly in the scale field. See Scaling the performance data chart display for more information.

Scaling the performance data chart display

You can manually adjust the scale for each counter so that the graph allows meaningful comparisons of different counters. Follow these steps to manually adjust the scale:

1. Double-click the **Scale** column for the counter that you want to modify.
2. Type the desired value in the field for the **Scale** value.

   The **View Chart** display immediately reflects the change in the scaling factor.

The possible values for the **Scale** field range from 0 to 100 and show the following relationships:

- A value equal to 1 indicates that the value is the actual value.
- A value greater than 1 indicates that the variable value is amplified by the factor shown. For example, a scale setting of 1.5 means that the variable is graphed as one and one-half times its actual value.

- A value less than 1 indicates that the variable value is decreased by the factor shown. For example, a scale setting of .5 means that the variable is graphed as one-half its actual value.

Scaling only applies to the graphed values.

Refreshing data

The refresh operation is a local, not global, operation that applies only to selected resources. The refresh operation is recursive; all subordinate or children resources refresh when a selected resource refreshes. To refresh data:

1. Click one or more resources in the Resource Selection panel.
2. Click **File** > **Refresh**. Alternatively, click the **Refresh** icon or right-click the resource and select **Refresh**. Clicking refresh with server selected under the viewer icon causes TPV to query the server for new PMI and product configuration information. Clicking refresh with server selected under the advisor icon causes TPV to refresh the advice provided, but will not refresh PMI or product configuration information.

Performance data refresh behavior: New performance data can become available in either of the following situations:

- An administrator uses the console to change the instrumentation level for a resource (for example, from medium to high).
- An administrator uses the console to add a new resource (for example, an enterprise bean or a servlet) to the run time.

In both cases, if the resource in question is already polled by the Tivoli Performance Viewer or the parent of the resource is being polled, the system is automatically refreshed. If more counters are added for a group that the performance viewer is already polling, the performance viewer automatically adds the counters to the table or chart views. If the parent of the newly added resource is polled, the new resource is detected automatically and added to the Resource Selection tree. You can refresh the Resource Selection tree, or parts of it, by selecting the appropriate node and clicking the **Refresh** icon, or by right-clicking a resource and choosing **Refresh**.

When an application server runs, the performance viewer tree automatically updates the server local structure, including its containers and enterprise beans, to reflect changes on the server. However, if a stopped server starts *after* the performance viewer starts, a manual refresh operation is required so that the server structure accurately reflects in the Resource Selection tree.

Clicking refresh with server selected under the viewer icon causes TPV to query the server for new PMI and product configuration information. Clicking refresh with server selected under the advisor icon causes TPV to refresh the advice provided, but will not refresh PMI or product configuration information.

Clearing values from tables and charts

Selecting **Clear Values** removes remaining data from a table or chart. You can then begin populating the table or chart with new data.

To clear the values currently displayed:

1. Click one or more resources in the Resource Selection panel.
2. Click **Setting** > **Clear Buffer**. Alternatively, right-click the resource and select **Clear Buffer**

Storing data to a log file

You can save all data reported by the Tivoli Performance Viewer in a log file and write the data in binary format (serialized Java objects) or XML format.

To start recording data:

1. Click **Logging** > **On** or click the **Logging** icon.
2. Specify the name, location, and format type of the log file in the **Save** dialog box. The **Files of type** field allows an extension of `*.perf` for binary files or `*.xml` for XML format.

   Note: The `*.perf` files may not be compatible between fix levels.
3. Click **OK**.

To stop logging, click **Logging** > **Off** or click the **Logging** icon.

Performance data log file: An example of the performance data log file format is below.

Location

By default, this file is written to:

product_installation_root`/logs/ra_mmdd_hhmm.xml`

where mmdd=month and date, and hhmm=hour and minute

Usage Notes

This read-write data file is created by Tivoli Performance Viewer and provides data collected by the performance viewer. The log file is not updated, but remains available for you to replay the collected data. The performance data log file does not have an effect on the WebSphere environment.

Example

```
<?xml version="1.0"?>
<RALog version="5.0">
<RAGroupSnapshot time="1019743202343" numberGroups="1">
  <CpdCollection name="root/peace/Default Server/jvmRuntimeModule" level="7">
   <CpdData name="root/peace/Default
            Server/jvmRuntimeModule/jvmRuntimeModule.total/Memory" id="1">
    <CpdLong value="39385600" time="1.019743203334E12"/>
   </CpdData>
   <CpdData name="root/peace/Default
         Server/jvmRuntimeModule/jvmRuntimeModule.freeMemory" id="2">
    <CpdLong value="4815656" time="1.019743203334E12"/>
   </CpdData>
   <CpdData name="root/peace/Default
        Server/jvmRuntimeModule/jvmRuntimeModule.usedMemory" id="3">
    <CpdLong value="34569944" time="1.019743203334E12"/>
   </CpdData>
  </CpdCollection>
 </RAGroupSnapshot>
</RALog>
```

Replaying a performance data log file

You can replay both binary and XML logs by using the Tivoli Performance Viewer.

To replay a log file, do the following:

1. Click **Data Collection** in the navigation tree.
2. Click the **Log** radio button in the **Performance data from** field.
3. Click **Browse** to locate the file that you want to replay or type the file path name in the **Log** field.
4. Click **Apply**.
5. Play the log by using the **Play** icon or click **Setting** > **Log Replay** > **Play**.

By default, the data replays at the same rate it was collected or written to the log. You can choose **Fast Forward** mode in which the log replays without simulating the refresh interval. To **Fast Forward**, use the button in the tool bar or click **Setting** > **Log Replay** > **FF**.

To rewind a log file, click **Setting** > **Log Replay** > **Rewind** or use the **Rewind** icon in the toolbar.

While replaying the log, you can choose different groups to view by selecting them in the Resource Selection pane. You can also view the data in either of the views available in the tabbed Data Monitoring panel.

You can stop and resume the log at any point. However, you cannot replay data in reverse.

Resetting counters to zero

Some counters report relative values based on how much the value has changed since the counter was enabled. The **Reset to Zero** operation resets those counters so that they will report changes in values since the reset operation. This operation will also clear the buffer for the selected resources. See ″Clearing values from tables and charts″ in Related Links for more information about clearing the buffer for selected resources. Counters based on absolute values can not be reset and will not be affected by the **Reset to Zero** operation.

To reset the start time for calculating relative counters:

1. Click one or more resources in the Resource Selection panel.
2. Click **Setting** > **Reset to Zero**. Alternatively, right-click the resource and click **Reset to Zero**.

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.

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 “Developing your own monitoring application with the Java Management Extension interface” on page 55. The PMI client interface is a Java interface. The servlet interface is perhaps the simplest, requiring minimal programming, as the output is XML.

1. "Developing your own monitoring application using Performance Monitoring Infrastructure client" on page 39.
2. "Developing your own monitoring applications with Performance Monitoring Infrastructure servlet" on page 52.
3. "Compiling your monitoring applications" on page 70.
4. "Running your new monitoring applications" on page 70.
5. "Developing your own monitoring application with the Java Management Extension interface" on page 55.
6. "Developing Performance Monitoring Infrastructure interfaces" on page 69.

Performance Monitoring Infrastructure client interface

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

Developing your own monitoring application using Performance Monitoring Infrastructure client

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.

Example: Performance Monitoring Infrastructure client with new data structure:

The following is example code using Performance Monitoring Infrastructure (PMI) client with the new 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 in 5.0) 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[]{jvmOName, ejbOName});

                // 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(wlmOName, 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(serverOName);

                // how to use update method
                testUpdate(jvmOName, 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;
        }
    }


}
```

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[]{jvmOName, ejbOName});

                // 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(wlmOName, 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(serverOName);

                // how to use update method
                testUpdate(jvmOName, 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;
        }

    }


}
```

Developing your own monitoring applications with Performance Monitoring Infrastructure servlet

The performance servlet uses the Performance Monitor Interface (PMI) infrastructure to retrieve the performance information from WebSphere Application Server. This is the same infrastructure used by the Tivoli Performance Viewer and is subject to the same restrictions on the availability of data as the performance viewer.

The performance servlet uses the Performance Monitor Interface (PMI) infrastructure to retrieve the performance information from WebSphere Application Server.

The performance servlet `.ear` file `perfServletApp.ear` is located in the *`install_root`* directory.

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`*

The performance servlet provides performance data output as an XML document, as described by the provided document type definition (DTD). The output structure provided is called `leaves`. The paths that lead to the leaves provide the context of the data. See the topic ″Performance Monitoring Infrastructure (PMI) servlet″ for more information about the PMI servlet output.

Performance Monitoring Infrastructure servlet:

The Performance Monitoring Infrastructure (PMI) servlet 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 PMI 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 performance servlet 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. There are three types of leaves or output formats within the XML structure:
- PerfNumericInfo
- PerfStatInfo
- 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 the observation was collected (Java System.currentTimeMillis)
- uid--Specifies the PMI identifier for this 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 this 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>
```

When the performance servlet is first initialized, it retrieves the list of nodes and servers located 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
```

By default, the performance servlet collects all of the performance data across a WebSphere domain. 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 TradeApp server on all nodes, invoke the following URL:

  ```
  http://hostname/wasPerfTool/servlet/perfservlet?Server=TradeApp
  ```

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

  ```
  http://hostname/wasPerfTool/servlet/perfservlet?Node=rjones&Server=TradeApp
  ```
- **Module.**The servlet can limit the information it provides to a specific PMI module by using the module parameter. You can request multiple modules from the following Web site:

  ```
  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 TradeApp on the node rjones, invoke the following URL:

  ```
  http://hostname/wasPerfTool/servlet/perfservlet?Node=rjones&Server=TradeApp
  &Module=beanModule>
  ```

Developing your own monitoring application with the Java Management Extension interface

WebSphere Application Server allows you to 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. PMI can be delivered in a single JMX cell through PerfMBean, 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.

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:

```
- setInstrumentationLevel: set the instrmentation level
            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
            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);
```

- **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 following is 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 to use AdminClient API directly to get PMI data from PerfMBean
 * and individual MBeans which support getStats method.
 */

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();

        // 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 = "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 = "8880";
        }
        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);
    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();
    }

}

/**
 * Some sample code to set level, get data, and display data.
 */
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(perfOName, "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(serverOName, 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
 */
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
 */
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(perfOName, "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(perfOName, "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.
     */
```

```
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;
    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.getName());
        if(config != null)
            oneStat.setConfig(config);
        else
            System.out.println("Error: get null config for " + oneStat.getName());
    }
}

/**
 * sample code to show how to get a specific MBeanStatDescriptor
 */
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
 */
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++)
        {
            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
 */
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
 */
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
     */
    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.
     */
    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(jvmOName,
                                                        "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, jvmOName));

                // 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();
        }
    }
}
```

Developing Performance Monitoring Infrastructure interfaces

This section discusses the use of the Performance Monitoring Infrastructure (PMI) client interfaces in applications. The basic steps in the programming model follow:

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);
   ```

Steps 2 and 3 are repeated through the lifetime of the client.

Compiling your monitoring applications

To compile your Performance Monitoring Infrastructure (PMI) code, you must have the following JAR files in your classpath:

- admin.jar
- wsexception.jar
- jmxc.jar
- pmi.jar
- pmiclient.jar
- ras.jar
- wasjmx.jar
- j2ee.jar
- soap.jar
- soap-sec.jar
- nls.jar
- ws-config-common.jar
- namingclient.jar

If your monitoring applications use APIs in other packages, also include those packages on the classpath.

Running your new monitoring applications

Follow these steps to run your monitoring applications.

1. Obtain the `pmi.jar` and `pmiclient.jar` files. The `pmi.jar` and `pmiclient.jar` files are required for client applications using Performance Monitoring Infrastructure (PMI) client APIs. The `pmi.jar` and `pmiclient.jar` files are distributed with WebSphere Application Server and are also a part of the Java thin client package. You can get these JAR files from either a WebSphere Application Server installation or a Java Thin Application Client installation. You also need the other JAR files in the Java Thin Application Client installation 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 classpath.
4. Run the application with the following script:

   ```
   call "%~dp0setupCmdLine.bat"

   set WAS_CP=%WAS_HOME%\properties
   set WAS_CP=%WAS_CP%;%WAS_HOME%\lib\pmi.jar
   set WAS_CP=%WAS_CP%;%WAS_HOME%\lib\pmiclient.jar
   set WAS_CP=%WAS_CP%;%WAS_HOME%\lib\ras.jar
   set WAS_CP=%WAS_CP%;%WAS_HOME%\lib\admin.jar
   ```

```
set WAS_CP=%WAS_CP%;%WAS_HOME%\lib\wasjmx.jar
set WAS_CP=%WAS_CP%;%WAS_HOME%\lib\j2ee.jar
set WAS_CP=%WAS_CP%;%WAS_HOME%\lib\soap.jar
set WAS_CP=%WAS_CP%;%WAS_HOME%\lib\soap-sec.jar
set WAS_CP=%WAS_CP%;%WAS_HOME%\lib\nls.jar
set WAS_CP=%WAS_CP%;%WAS_HOME%\lib\wsexception.jar
set WAS_CP=%WAS_CP%;%WAS_HOME%\lib\ws-config-common.jar
set WAS_CP=%WAS_CP%;%WAS_HOME%\lib\namingclient.jar

%JAVA_HOME%\bin\java "%CLIENTSOAP%" "%CLIENTSAS%" "-Dws.ext.dirs=%WAS_EXT_DIRS%"
%DEBUGOPTS% -classpath "%WAS_CP%" com.ibm.websphere.pmi.PmiClientTest host name
[port] [connectorType]
```

Performance Monitoring Infrastructure client package:

A Performance Monitoring Infrastructure (PMI) client package provides a PmiClient wrapper class to deliver PMI data to a client.

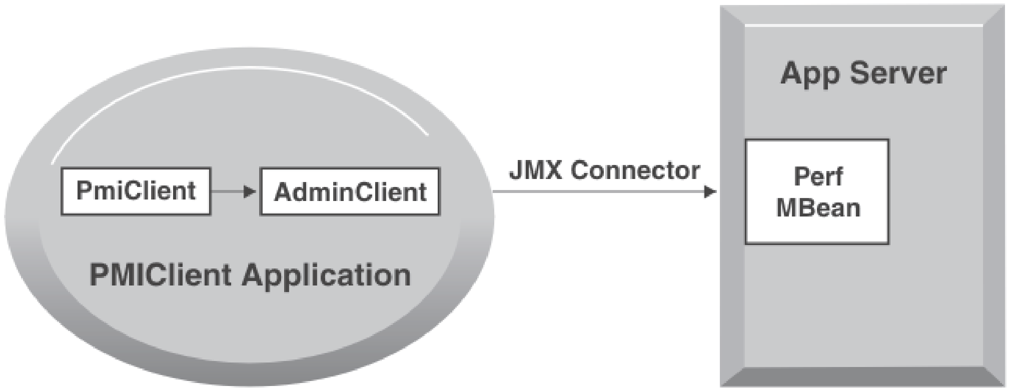
As shown in the following figure, the PmiClient API uses the AdminClient API to communicate to the Perf MBean in an application server.

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:

In order to run a Performance Monitoring Infrastructure (PMI) client application with security enabled, you must have %CLIENTSOAP% and %CLIENTSAS% properties on your Java virtual machine command line. The %CLIENTSOAP% and %CLIENTSAS% properties are defined in the `setupCmdLine.bat` or `setupCmdline.sh` files.

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 `WAS_ROOT/properties` directory.
2. Set `com.ibm.SOAP.loginUserid` and `com.ibm.SOAP.loginPassword` as the user ID and password for login.
3. 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.

Third-party performance monitoring and management solutions

Several other companies provide performance monitoring, problem determination, and management solutions that can be used with WebSphere Application Server.

These products use WebSphere Application Server interfaces, including Performance Monitoring Infrastructure (PMI), Java Management Extensions (JMX), and PMI Request Metrics Application Response Measurement (ARM).

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 87 for a link to IBM business partners providing monitoring solutions for WebSphere Application Server.

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__
```

Steps for capturing a workload activity report

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(O)
    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.

WLM Delay Monitoring

WebSphere Application Server for z/OS Version 5 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 1. 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	Called a J2C connector to perhaps DB2, CICS, IMS. Investigate 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 may 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.

Table 1. WLM delay monitoring states (continued)

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.
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 in 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

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
```

Measuring data requests (Performance Monitoring Infrastructure Request Metrics)

Performance Monitoring Infrastructure (PMI) Request Metrics is a tool that allows you to track individual transactions, recording the processing time in each of the major WebSphere Application Server components. The information tracked may either be saved to log files for later retrieval and analysis, be sent to ARM Agents, or both.

As a transaction flows through the system, Request Metrics tacks on 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:

```
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 users target performance problem areas and debug resource constraint problems. For example, the flow can help determine if a transaction is spending most of its time in the Web server plug-in, the Web container, the enterprise bean container or the backend database. The response time 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 EJB and JDBC. Therefore, 92ms can be attributed to the servlet process.

Request metrics tracks the response time for a desired transaction. For example, tools can inject synthetic transactions. Request Metrics can then track the response time within the WebSphere environment for those transactions. A synthetic transaction is one that is injected into the system by administrators in order to proactively test the performance of the system. This information can help administrators tune the performance of the Web site and take corrective actions should they be needed. Thus, the information provided by Request Metrics might be used as an alert mechanism to detect when the performance of particular request type goes beyond acceptable thresholds. The filtering mechanism within Request Metrics may be used to focus on the specific synthetic transactions and can help optimize performance in this scenario.

Three types of filters are supported:
- Originator IP filter
- URI filter
- EJB method name filter

When filtering is enabled, only requests matching the filter generate Request Metrics data, create log records, and/or call the ARM interfaces. This allows work to be injected 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 be hitting the system.

Learn more about Request Metrics by reviewing this section, including:
- Detailed explanation about Request Metrics
- Request Metrics process and filters
- Types and format of output you will be reading
- "Configuring Request Metrics" on page 81

Performance Monitoring Infrastructure Request Metrics

Typically, there are multiple processes across several nodes in a distributed system. When a request comes to a process, the process may send the request to one or more downstream processes.

Trace records may be generated for each process with associated elapsed times for that process. These trace records may be correlated together to build a complete picture of the request flow through the distributed system, similar to the diagram in "Measuring data requests (Performance Monitoring Infrastructure Request Metrics)" on page 77.

The process response time monitored by Request Metrics can be viewed through the Application Response Measurement (ARM) interface and system logs. For requests that originate from either an HTTP request or the remote interface of an enterprise bean, Request Metrics captures response times for the initiating request and any related downstream invocations. If the request originated as an HTTP request, response times are generated for web server plug-in (only available when using web server port), the web container, the EJB container, and JDBC calls. If the request originated as a remote EJB call, response times are generated for the EJB container and JDBC calls. Note that the JDBC response time are only traced for the WebSphere 5.0 data source. No response time will be traced for WebSphere 4.0 data source.Depending on the trace level configured for request metrics, the intra-process servlet and EJB calls may or may not be traced.

When active, Request Metrics compares each incoming request to a set of known filters. Three types of filters are supported:
- Originator IP filter
- URI filter
- EJB method name filter

When filtering is enabled, only requests matching the filter generate Request Metrics data, create log records, and/or call the ARM interfaces. This allows work to be injected 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 be hitting the system. If the request matches any filter with a trace level greater than TRACE_NONE, trace records are generated for that request.

Trace records are generated and logged for the Web Server plug-in, servlets (WebContainer), remote enterprise bean calls, and Java Database Connectivity (JDBC drivers).

Application Response Measurement

Application Response Measurement (ARM) is an Open Group standard composed of a set of interfaces implemented by an ARM agent that provides information on elapsed time for process hops.

WebSphere Application Server does not provide an ARM agent. Contact your ARM agent provider for information on whether their ARM agent is supported with WebSphere Application Server. One product that uses ARM agents is described in Tivoli performance monitoring and management solutions

See the article “Performance: Resources for learning” on page 87 for more information about the ARM specifications.

Performance Monitoring Infrastructure Request Metrics trace filters

When Performance Monitoring Infrastructure (PMI) Request Metrics is active, trace filters control which requests get traced. The data is recorded to the system log file or sent through ARM for real-time and historical analysis.

Incoming HTTP requests

HTTP requests arriving at the WebSphere Application Server may be filtered based on the URI and/or the IP address of the originator of the request.
- **Originator 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.
- **URI filters.** Requests are filtered, based on the URI of the incoming HTTP request. The rules for pattern matching are the same as for matching Originator IP address filters.
- **Filter combinations.**If both URI and Originator IP address filters are active, then Request Metrics requires a match for both filter types. If neither is active, all requests are considered a match.

Incoming enterprise bean requests
- **Enterprise bean method name filters.** Requests are filtered based on the full name of the enterprise bean method. As with IP address and URI filters, the asterisk (*) may 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.

Performance Monitoring Infrastructure Request Metrics data output

The trace records for Performance Monitoring Infrastructure (PMI) Request Metrics data are output to two log files: the Web server plug-in log file and the application server log file. The default directory for these log files is `<$WAS_ROOT/logs>` (or the name given to your server `<$WAS-ROOT/logs/`*server_name*`>`) and default names are `SystemOut.log` and `http_plugin.log`. Users might, however, specify these log file names and their locations.

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 occurred as it entered 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. The logical tree can be created 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 “Measuring data requests (Performance Monitoring Infrastructure Request Metrics)” on page 77 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 as follows:
- **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.
- **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 assigned to the request by Request Metrics, unique to the application server process.
- **event:** An event ID assigned to differentiate the actual trace events.

Following the parent and current correlators, is the metrics data for timed operation:
- **type**: A code representing the type of operation being timed. Supported types include `HTTP`, `URI`, `EJB` and `JDBC`.
- **detail**: Identifies the name of the operation being timed (See the description of Universal Resource Identifier (URI), `HTTP`, Enterprise bean and Java Database Connectivity (JDBC) below.)
- **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 received by the Web server plug-in.
- **`bytesOut`**: The number of bytes from the reply sent from the Web server plug-in to the client.

The type and detail fields are described as follows:
- **`HTTP`**: The Web server plug-in generates the trace record. The detail is the name of the URI used to invoke the request.
- **`URI`**: The trace record was generated by a Web component. The URI is the name of the URI used to invoke the request.
- **`EJB`**: The fully qualified package and method name of the enterprise bean.
- **`JDBC`**: The values select, update, insert or delete for prepared statements. For non-prepared statements, the full statement can appear.

Configuring Request Metrics

You can enable Request Metrics without enabling Application Response Measurement (ARM).

To configure Request Metrics, you will need to access the Configuration tab in the administrative console. To access the Configuration tab , click **Troubleshooting** > **PMI Request Metrics** from the administrative console navigation tree.

Tasks included in configuring Request Metrics:
1. "Enabling Performance Monitoring Infrastructure Request Metrics" on page 85.
2. "Enabling Application Response Measurement" on page 86.
3. Enabling Performance Monitoring Infrastructure Request Metrics filters.
4. Adding and removing Performance Monitoring Infrastructure Request Metrics filters.
5. Setting the trace level in Performance Monitoring Infrastructure Request Metrics.
6. Regenerating the Web server plug-in configuration file.

Performance Monitoring Infrastructure Request Metrics

Use this page to enable Performance Monitoring Infrastructure (PMI) Request Metrics, enable Request Metrics Application Response Measurement (ARM), and set trace levels.

To view this administrative console page, click **Troubleshooting** > **PMI Request Metrics**.

Request Metrics:

Enables PMI Request Metrics.

When disabled, the Request Metrics function is disabled.

Application Response Measurement (ARM):

Enables PMI Request Metrics to call an underlying ARM agent.

Before enabling ARM, install an appropriate ARM implementation on all WebSphere Application Server nodes. Verify with your ARM agent provider that Request Metrics is supported by the ARM agent implementation. ARM support is dependent on Request Metrics support.

Trace Level:

Specifies how much trace data to accumulate for a given request.

Including one of the following:

NONE
: No trace.

HOPS
: Only accumulates on major process hops.

PERF_DEBUG
: Reserved for enabling additional information over hops, but is not as performance intensive as DEBUG. Currently, provides the same level of tracing as HOPS.

PERF_DEBUG
: Reserved for enabling additional information over hops, but is not as performance intensive as DEBUG. Starting from 5.1.1, it enables response time monitoring for intra-process servlet and EJB calls that are called from a process inbound servlet or EJB.

DEBUG
: Reserved for full detailed trace. Currently, provides the same level of tracing as HOPS.

DEBUG
: Reserved for full detailed trace. Starting from 5.1.1, it enables response time monitoring for all the intra-process servlet and EJB calls.

Enabling and disabling logging:

Need a statement here of anything user needs to have or know before following these steps.

You can enable and disable the logging by setting a JVM custom property called com.ibm.websphere.pmi.reqmetrics.loggingEnabled.

If this property is set, the request metrics will enable and disable logging based on its value. If this property is not set, request metrics will enable logging only when the trace level is set to something other than NONE. When trace level is set to none, logging is disabled no matter if the property is set. Since the trace level is also used to control the monitoring for the ARM agent, you should disable logging via the system property when ARM is enabled.

To set the JVM custom property, do the following:

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

3. Click on **server1** on the Application Servers page and select **Process Definition** from the Additional properties menu on the server1 page.
4. Click on **Java Virtual Machine** from the Additional properties menu on the Process Definition page.
5. Click on **Custom Properties** from the Additional properties menu on the Java Virtual Machine page.
6. Click on **New** and enter the custom property **com.ibm.websphere.pmi.reqmetrics.loggingEnabled** in the General Properties Name field and enter a value of either true (to enable) or false (to disable) under the Configuration tab.
7. Click **Apply** or **OK**.
8. Click **Save**.
9. Restart the application server for this change to take effect.

PMIRM Filter collection:

Use this page to view a list of Performance Monitoring Infrastructure (PMI) Request Metrics filters.

To view this administrative console page, click **Troubleshooting** > **PMI Request Metrics** > **Filters**.

Type:

Specifies the type of request metrics filter.

Enable:

Specifies whether this filter is enabled. Note: this has to be enabled in order to enable the filter values under this filter type.

PMIRM Filter settings:

Use this page to specify filters that define whether or not trace is enabled for the request as it moves through WebSphere Application Server.

To view this administrative console page, click **Troubleshooting** > **PMI Request Metrics** > **filters** > *filter_type*.

Regenerate the web server plugin configuration file after saving the changes if you want to use request metrics in web server plugin. Although the changes are made under the Configuration tab, after the changes are saved, they will take immediate effect in server runtime. This applies to all the changes under request metrics.

Type:

Specifies the type of Request Metrics filter.

Enable:

Specifies whether this filter is enabled.

filterValues:

Specifies a filter's value and enablement for the filter type.

filterValues collection:

Use this page to specify the values for client IP, URI or EJB Request Metrics filters.

To view this administrative console page, click **Troubleshooting** > **PMI Request Metrics** > **filters** > *filter_type* > **filterValues**.

Value:

Specifies a URI value or IP name 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.

filterValues settings:

Use this page to specify the values for client IP, URI or EJB Request Metrics filters.

To view this administrative console page, click **Troubleshooting** > **PMI Request Metrics** > **filters** > *filter* > **filterValues** > *filter_value*.

Value:

Specifies a URI value or IP name based on the type of filter.

For example, for URI filters, the value can be ″/servlet/snoop″.

Enable:

Specifies whether this filter value is enabled.

Example: Setting Custom JVM properties to choose an ARM agent: Once you enable ARM, you need to set the proper JVM custom properties for the type of ARM agent and the ARM transaction factory. name is ArmTransactionFactory, whose

For the type of ARM agent, the property is:

com.ibm.websphere.pmi.reqmetrics.ARMIMPL
: Specifies the type of ARM agent.

  The value can be: ARM4, eWLM, or Tivoli.

  Steps for this task

  1. Connect to the administrative console and navigate to the Java Virtual Machine Custom Properties panel. For example, **Servers > Application Servers >***server1* **> Process Definition > > Java Virtual Machine > Custom Properties**
  2. If the **com.ibm.websphere.pmi.reqmetrics.ARMIMPL** property is not present in the list, create a new property name.
  3. Enter the name and value.

For the ARM transaction factory, the property is:

ArmTransactionFactory

The value should be the ArmTransactionFactory implementation class name (need to start from the package name).

Steps for this task

1. Connect to the administrative console and navigate to the Java Virtual Machine Custom Properties panel. For example, **Servers > Application Servers >***server1* **> Process Definition > > Java Virtual Machine > Custom Properties**
2. If the **ArmTransactionFactory** property is not present in the list, create a new property name.
3. Enter the name and value.

Note: You must restart the application server in order for this change to take effect. In a Network Deployment installation, you should also set the property for the deployment manager. If you do not set it in deployment manager, you may see two problems:

1. The plugin configuration regeneration cannot set the correct loggingEnable value in the plugin configuration file.
2. When migrated to 6.0, the property cannot be picked up.

Example: Setting Custom JVM properties to enable or disable logging: You can enable and disable the logging by setting a JVM custom property called:

com.ibm.websphere.pmi.reqmetrics.loggingEnabled

If this property is set, the request metrics will enable and disable logging based on its value. If this property is not set, the logging will be enabled when the trace level is set to something other than NONE. When trace level is set to none, logging will be disabled no matter if the property is set. Since the trace level is also used to control the monitoring for ARM agent, it is recommended that you disable logging via the system property when ARM is enabled.

The value can be set to true or false.

Steps for this task

1. Connect to the administrative console and navigate to the Java Virtual Machine Custom Properties panel. For example, **Servers > Application Servers >***server1* **> Process Definition > > Java Virtual Machine > Custom Properties**.
2. If the **com.ibm.websphere.pmi.reqmetrics.loggingEnabled** property is not present in the list, create a new property name.
3. Enter the name and value.

Enabling Performance Monitoring Infrastructure Request Metrics

When enabled, Performance Monitoring Infrastructure (PMI) Request Metrics captures response times for the initiating request and any related downstream enterprise bean invocations and Java Database Connectivity (JDBC) calls. Then, Request Metrics compares each incoming request to a set of known filters.

1. Open the administrative console.
2. Click **Troubleshooting** > **PMI Request Metrics** in the console navigation tree.
3. Select the check box in the **enable** field under the Configuration tab.
4. Click **Apply** or **OK**.
5. Click **Save**.

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, if logging time spent in the Web server.

Enabling Application Response Measurement

Before enabling Application Response Measurement (ARM), install an appropriate ARM implementation on all WebSphere Application Server nodes. Refer to the appropriate ARM implementation documentation. Verify with your ARM agent provider that the ARM agent is supported with the request metrics in WebSphere.

You can learn more about ARM agents in the topic "Performance: Resources for learning" on page 87.

Note: Request Metrics in the Web server plug-in is not integrated with ARM in WebSphere Application Server Version 5.0.x. Therefore, Request Metrics in the Web Server plug-in ignores ARM, if enabled.

1. Install the appropriate ARM implementation
   a. Change the startup command for the application servers to include the following:
      ```
      -Dcom.ibm.websphere.pmi.reqmetrics.ARMIMPL=ARMIMPLNAME
      ```
      ARM support is dependent on Request Metrics support. If enabled, and an appropriate ARM implementation is defined to the server run times, then the ARM implementation is called as requests enter WebSphere Application Server processes and when Java Database Connectivity (JDBC) calls are made, using EJB 2.0 data sources.
2. Open the administrative console.
3. Click **Troubleshooting** > **PMI Request Metrics** in the console navigation tree.
4. Select the check box in the **enableARM** field.
5. Click **Apply** or **OK**.
6. Click **Save**.

Regenerating the Web server plug-in configuration file.

Correlation service interface

The only new API is for ARM transaction stack access.

The API is organized as:

- An ARM Transaction wrapper object factory.
- A set of ARM Transaction wrapper object types implementing the PmiRmArmTx interface.
- An ARM transaction stack manipulator class.

The PmiRmArmTxFactory exposes a create method (createPmiRmArmTx) to create the correct type of ARM transaction object depending on the environment.

```
public class PmiRmArmTxFactory {
 public static int EWLM_TYPE = 0;
 public static int TIVOLI_TYPE = 1;
 public static int ARM4_TYPE = 2;
 public static PmiRmArmTx createPmiRmArmTx();
}
```

The PmiRmArmTxFactory create method returns the PmiRmArmTx interface type. The interface provides mechanisms to set and get the correlator byte array into and from the ARM transaction object, and exposes various ARM methods which it delegates to the underlying real ARM transaction. Depending on the type of underlying ARM transaction, some methods may or may not be implemented. For instance, if the ARM type is TIVOLI_TYPE, the blocked and unblocked methods are not implemented since the TIVOLI wrappers are based on Tivoli's ARM2 agent which does not support blocked/unblocked.

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 links to additional information about:
- Monitoring performance with third-party tools
- Tuning performance
- Garbage collection

Monitoring performance with third-party tools
- Enterprise Web Application Management WebSphere Performance Management Business Partner Solution Finder

  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
- Application tuning

  See WebSphere Application Server Development Best Practices for Performance and Scalability for more information on application tuning.
- Performance Analysis for Java Web sites
- WebSphere Application Server Development Best Practices for Performance and Scalability

  Describes development best practices for Web applications with servlets, JSP files, JDBC connections, and enterprise applications with EJB components.

Garbage collection
- 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.

- Tuning Garbage Collection with the 1.3.1 JavaTM Virtual Machine

  Learn more about using garbage collection in a Solaris operating environment.

Chapter 3. Tuning performance parameter index

Tuning parameter index for z/OS

Performance tuning for WebSphere for z/OS becomes a complex exercise because the nature of the runtime involves many different components of the operating system and middleware. Use the "Tuning index for WebSphere Application Server for z/OS" on page 90 to find information and parameters for tuning the z/OS operating system, subsystems, the WebSphere for z/OS runtime environment, and some J2EE application tuning tips.

Recommendation: Before you read a description of WebSphere for z/OS tuning guidelines, it is important to note that, no matter how well the middleware is tuned, it cannot make up for poorly designed and coded applications. Focusing on the application code can help improve performance. Often, poorly written or designed application code changes will make the most dramatic improvements to overall performance.

The tuning guide focuses on server tuning. If you want to tune your applications, see "Performance: Resources for learning" on page 87 for more information about application tuning.

For your convenience, procedures for tuning parameters in other products, such as DB2, Web servers and operating systems are included. Because these products might change, consider these descriptions as suggestions.

Each WebSphere Application Server process has several parameters influencing application performance. You can use the WebSphere Application Server administrative console to configure and tune applications, Web containers, EJB containers, application servers and nodes in the administrative domain.

If you are a WebSphere Application Server Administrator or Systems programmer on z/OS, refer to "Tuning index for WebSphere Application Server for z/OS" on page 90 for z/OS specific tuning tips.

Each parameter in the tuning parameter index links to information that explains the parameter, provides reasons to adjust the parameter, how to view or set the parameter, as well as default and recommended values.

- Tuning application servers

  The WebSphere Application Server contains interrelated components that must be harmoniously tuned to support the custom needs of your end-to-end e-business application.
- Tuning Java virtual machines

  The JVM offers several tuning parameters affecting the performance of WebSphere Application Servers (which are primarily Java applications), as well as the performance of your applications.
- "Tuning applications" on page 131

  Several topics including Web modules, EJB modules, client modules, Web services and application services comprise the application programming model and provide numerous services supporting deployed applications.
- Tuning databases

WebSphere supports the integration of several different database systems. Each is tuned in its own manor. DB2 tuning parameters are provided for your convenience.
- Tuning Java messaging service

  Java messaging service (JMS) can be tuned to balance memory with the servicing of the JMS subscribers.
- Tuning security

  Security may have an affect on performance depending on certain actions taken.

Recommended hardware configuration

The tuning guidelines presented in this chapter will have the most benefit on the following recommended configuration:
- An IBM S/390 or zSeries Model that supports the software requirement of z/OS V1R2.
- Storage

  Storage requirements are higher than for traditional workloads
  - Virtual storage usage by WebSphere for z/OS and its applications varies depending on your configuration. In the simplest configuration (a daemon, a controller, and a servant region), plan for a minimum of 500 MB of virtual. Virtual storage requirements can be much larger in more complex configurations.
  - Real storage minimum is 384 MB per LPAR for a light load such as the IVP. For most real-world applications, we recommend 2 GB or higher. We have seen applications that require as much as 8 GB of real storage to operate at peak load.

  For information on tuning storage, refer to “Tuning storage” on page 91.
- DASD

  To maximize your performance, we recommend a fast DASD subsystem (for example, IBM Shark), running with a high cache read/write hit rate.
- Networking

  For high bandwidth applications, we recommend at least a 1 Gb Ethernet connection. If your applications have extremely high bandwidth requirements, you may need additional Ethernet connections.
- Cryptography

  For applications that make heavy use of cryptography, we recommend the zSeries or S/390 cryptographic hardware and the Integrated Cryptographic Service Facility. For more information, refer to the zSeries and S/390 Cryptography Web site.

Tuning index for WebSphere Application Server for z/OS

One of the goals of the WebSphere Application Server for z/OS programming model and runtime is to significantly simplify the work required for application developers to write and deploy applications. Sometimes we say that WebSphere Application Server for z/OS relieves the application programmer of many of the plumbing tasks involved in developing applications. For example, application code in WebSphere Application Server for z/OS does not concern itself directly with remote communication--it locates objects which may be local or remote and drives methods. Therefore, you won’t see any direct use of socket calls or TCP/IP programming in a WebSphere Application Server for z/OS application.

This separation of what you want to do from where you do it is one aspect of removing the application programmers from plumbing tasks. Other considerations are not having to deal with data calls for some types of beans, potentially user authentication, and threading. There are generally no calls from the application code to touch sockets, RACF calls, or management of threading. Removing this from the application programmer doesn't mean this work won't get done. Rather, it means that there may be more work for the DBA, the network administrator, the security administrator, and the performance analyst.

There are four layers of tuning that need to be addressed:
- "Tuning the z/OS operating system"
- "Tuning for subsystems" on page 101
- "Tuning the WebSphere Application Server for z/OS runtime" on page 114
- "Tuning for J2EE applications" on page 123

We deal with the first three in separate sections under this article and briefly touch on the fourth. For more information on tuning applications, refer to Using application clients.

Tuning the z/OS operating system

Steps involved in tuning the z/OS operating system to optimize WebSphere performance include:
- "Tuning storage"
- "z/OS or OS/390 operating system tuning tips" on page 93
- "Resource Recovery Service (RRS) tuning tips for z/OS" on page 94
- "UNIX System Services (USS) tuning tips for z/OS" on page 99
- "Workload management (WLM) tuning tips for z/OS" on page 100

Tuning storage

WebSphere for z/OS puts much higher demands on virtual memory than a traditional workload. Ensure that you don't underestimate the amount of virtual storage applied to the WebSphere for z/OS servers. Generally, they use significantly more virtual memory than traditional application servers on z/OS or OS/390. Since real storage is needed to back the virtual storage, real storage usage may also be high.

1. Allocate enough virtual storage. The setting of REGION on the JCL for the proc should be large (at least 200MB to run), and much larger if high throughput is required. You can get an idea of the virtual storage usage through RMF or other performance monitors. It would not be unreasonable for the servant procs to specify REGION=0M, which tells the operating system to give all the available region (close to 2GB).

   Note: For more information on REGION=0M and IEFUSI, please see "Preparing the base OS/390 or z/OS environment" in the Getting Started section of the information center.

   If you choose to not put most of the runtime in LPA, as described in the program locations section, be sure to specify more region (as high as 512MB). Also, in conjunction with the increase in storage usage, you may have to define more paging space or auxiliary storage to back up the additional virtual storage used.
2. Optional: Allocate enough real storage. Expect a requirement of at least 384MB of real storage for a small configuration. For controllers and servants, real storage utilizations depends on the size of the JVM heapsize.

   Recommendation: It can be the case that in a heavy use environment 2G of central storage is not enough to handle the real storage demands of a high volume Java application. In this case, we recommend that you configure with 64-bit real storage, which will give you the ability to dedicate more central storage to the LPAR. z/OS on a zSeries will always run in 64-bit mode. Running in 64-bit mode gives you the ability to define more than 2 GB of central storage When you configure for 64-bit real all of the storage is defined as central storage. For non-zSeries processors, or 31-bit mode, you can minimize paging by defining more expanded storage.

3. Tune the Java virtual machine storage. Also refer to Best practices for maintaining the run-time environment.

Java virtual machine storage tuning tips for z/OS: Specifying a sufficient `JVM Heap Size` is important to Java performance. The JVM has thresholds it uses to manage the JVM's storage. When the thresholds are reached, the garbage collector (GC) gets invoked to free up unused storage. GC can cause significant degradation of Java performance.

Use the administrative console to specify the Initial Heap Size and the Maximum Heap Size for the JVM.

To view this administrative console page, click **Servers > Application Servers** >*server_name* > **Process Definition > Java Virtual Machine**. Access the configuration tab to change these settings.

- In order to get it to run less frequently, you can give the JVM more memory. This is done by specifying a larger value for `Initial Heap Size`. The default of 256M is a good starting point but may need to be raised for larger applications. When specifying either a larger or smaller JVM heap size value, IBM recommends that you code **both** the initial and maximum values that you desire.
- It is good for the Initial Heap Size to equal the Maximum Heap Size because it allows the allocated storage to be completely filled before GC kicks in. Otherwise, GC will run more frequently than necessary, potentially impacting performance.
- Make sure the region is large enough to hold the specified JVM heap.
- Beware of making the `Initial Heap Size` *too* large. While it initially improves performance by delaying garbage collection, it ultimately affects response time when garbage collection eventually kicks in (because it runs for a longer time).
- Paging activity on your system must also be considered when you set your JVM heap size. If your system is already paging heavily, increasing the JVM heap size might make performance worse rather than better.
- To determine if you are being affected by garbage collection, you can enable Verbose Garbage Collection on the JVM Configuration tab. The default is not enabled. This will write a report to the output stream each time the garbage collector runs. This report should give you an idea of what is going on with Java GC.

  Example: This is an example of a verboseGC report.

  ```
  ..
  <AF[21]: Allocation Failure. need 32784 bytes, 32225 ms since last AF
  >
  <AF[21]: managing allocation failure, action=1 (84320/131004928)
                                                (3145728/3145728)>
  <GC(21): GC cycle started Wed Feb 27 22:46:11 2002
  <GC(21): freed 99587928 bytes, 76% free (102817976/134150656),
   in 118 ms>
  <GC(21): mark: 103 ms, sweep: 15 ms, compact: 0 ms>
  ```

```
<GC(21): refs: soft 0 (age >= 32), weak 0, final 878, phantom 0>
<AF[21]: completed in 118 ms
>.
..
```

Key things to look for in a verboseGC report are:

– Time spent in garbage collection.

  Ideally, you want to be spending less than 5% of the time in GC. To determine percentage of time spent in GC, divide the time it took to complete the collection by the time since the last AF and multiply the result by 100. For example,

  ```
  118/32225 * 100 = 0.366%
  ```

  If you are spending more than 5% of your time in GC and if GC is occurring frequently, you may need to increase your Java heap size.

– Growth in the allocated heap.

  To determine this, look at the %free. You want to make sure the number is not continuing to decline. If the %free continues to decline you are experiencing a gradual growth in allocated heap from GC to GC which could indicate that your application has a memory leak.

You can also use the MVS console command, *modify display, jvmheap* to display JVM heap information. See "Modify command" for details. In addition, you can check the server activity and interval SMF records. The JVM heap size is also made available to PMI and can be monitored using the Tivoli Performance Viewer.

z/OS or OS/390 operating system tuning tips

This section provides tuning tips for various components of z/OS or OS/390.

- CTRACE

  The first place to review is your CTRACE configuration. Ensure that all components are either set to MIN or OFF. To display the CTRACE options for all components on your system, issue the following command from the operator console:

  ```
  D TRACE,COMP=ALL
  ```

  To change the setting for an individual component to its minimum tracing value, use the following command (where 'xxx' is the component):

  ```
  TRACE CT,OFF,COMP=xxxx
  ```

  This will eliminate any unnecessary overhead of collecting trace information that is not being used. Often during debug, CTRACE is turned on for a component and not shut off when the problem is resolved.

- SMF

  Ensure that you are not collecting more SMF data than you need. Review the SMFPRMxx to ensure that only the minimum number of records are being collected.

  Use SMF 92 or 120 only for diagnostics.

  – SMF Type 92

    SMF Type 92 records are created each time an HFS file is opened, closed, deleted, and so forth. Almost every web server request references HFS files, so thousands of SMF Type 92 records are created. Unless you specifically need this information, turn off SMF Type 92 records. In the following example, we have disabled the collection of SMF type 92 records:

    Example:

    ```
    ACTIVE,
    DSNAME(SYS1.&.SYSNAME..SMF.MAN1;SYS1.&SYSNAME..SMF.MAN2;),
    NOPROMPT,
    ```

```
REC(PERM),
MAXDORM(3000),
STATUS(010000),
JWT(0510),
SID(&SYSNAME;(1:4)),
LISTDSN,
SYS(NOTYPE(19,40,92)),
INTVAL(30),
SYNCVAL(00),
SYS(DETAIL,INTERVAL(SMF,SYNC)),
SYS(EXITS(IEFACTRT,IEFUJI,IEFU29,IEFU83,IEFU84,IEFU85,IEFUJV,IEFUSI))
```

– SMF Type 120

  You may find that running with SMF 120 records in production is appropriate, since these records give information specific to WebSphere applications such as response time for J2EE artifiacts, bytes transferred, and so forth. If you do choose to run with SMF 120 records enabled, we recommend that you use server interval SMF records and container interval SMF records rather than server activity records and container activity records. For details of the SMF 120 record, refer to Record Type 120 (78) - WebSphere for z/OS performance statistics.

  For steps involved in controlling collection of SMF 120 records refer to Enabling SMF recording. To enable specific record types, specify the following properties:

  - server_SMF_server_activity_enabled=0 (or server_SMF_server_activity_enabled = false)
  - server_SMF_server_interval_enabled=1 (or server_SMF_server_activity_enabled = true)
  - server_SMF_container_activity_enabled=0 (or false)
  - server_SMF_container_interval_enabled=1 (or true)
  - server_SMF_interval_length=1800

- You might also want to review your DB2 records and the standard RMF written SMF records, and ensure that the SMF datasets are allocated optimally. Information on ensuring a high performance SMF recording environment can be found in the chapter on Customizing SMF in *z/OS MVS System Management Facilities (SMF)*.

Resource Recovery Service (RRS) tuning tips for z/OS

- For best throughput, use coupling facility (CF) logger.

  DASD logger can limit your throughput because it is I/O-sensitive. The CF logger has much more throughput (in one measurement, the CF logger was six times faster than the DASD logger). Throughput will benefit from moving the RRS logs in logger to a coupling facility (CF) logstream. Doing so will help transactions complete quickly and not require any DASD I/O. If it's not possible to use CF logs, use well performing DASD and make sure the logs are allocated with large CI sizes.

- Ensure that your logger configuration is optimal by using SMF 88 records.

  See the tuning section of *z/OS MVS Setting Up a Sysplex* or the chapter on System Logger Accounting in *z/OS MVS System Management Facilities (SMF)* for details. In any case, you should monitor the logger to ensure that there is a sufficient size in the CF and that offloading is not impacting the overall throughput. The transaction logs are one of the only shared I/O intensive resources in the mainline and can affect throughput dramatically if they are mistuned.

- Set adequate default values for the LOGR policy.

  Default values of LOGR policy may have an impact on performance. We recommend the default settings in the table below.

Table 2. Recommended default setting for LOGR

Log Stream	Initial Size	Size
RM.DATA	1 MB	1MB
MAIN.UR	5 MB	50 MB
DELAYED .UR	5 MB	50 MB
RESTART	1 MB	5 MB
ARCHIVE	5 MB	50 MB

- Review XA Resource Managers log sizes.

  If you are using XA Resource Managers and you have chosen to put the logs in the logger, you may have to review the log sizes. As of this writing, we cannot give specific recommendations.

  You can configure the XA logs in the install dialog to live either in the HFS or in logstreams. If you are not using global transactions involving XA resources, there is no point in putting the log in a logstream. If the XA logs are placed in logstreams, we recommend that they be in the Coupling Facility instead of DASD. The default names are ’HLQ.server.M’ and ’HLQ.server.D’ where HLQ is a user-defined value between 1-8 characters specified in the install dialog, and ’server’ is the server short name. It is the installer’s responsibility to ensure that the HLQ + server name is unique across the configuration. If it is not, the server will fail to start because the user data in the existing logstream will not match that of the new server. The logs (and structures, if applicable) are created in job ’BBOLOGSA’ in the install dialog. If structures need to be allocated, there is also a step indicating what structure names need to be added to the CFRM policy. We recommend 5MB initial and 20MB max sizes for both of these logstreams.
- Eliminate archive log if not needed.

  If you don’t need the archive log, we recommend that you eliminate it since it can introduce extra DASD I/Os. The archive log contains the results of completed transactions. Normally, the archive log is not needed. Following is an example of disabling archive logging.

  Example:

```
//STEP1 EXEC PGM=IXCMIAPU
//SYSPRINT DD  SYSOUT=*
//SYSIN    DD  *
    DATA TYPE(LOGR)
    DELETE LOGSTREAM NAME(ATR.WITPLEX.ARCHIVE)

    DELETE LOGSTREAM NAME(ATR.WITPLEX.MAIN.UR)
    DELETE LOGSTREAM NAME(ATR.WITPLEX.RESTART)
    DELETE LOGSTREAM NAME(ATR.WITPLEX.RM.DATA)
    DELETE LOGSTREAM NAME(ATR.WITPLEX.DELAYED.UR)
    DELETE STRUCTURE NAME(RRSSTRUCT1)
/*
//STEP2 EXEC PGM=IXCMIAPU
//SYSPRINT DD  SYSOUT=*
//SYSIN    DD  *
    DATA TYPE(LOGR)
    DEFINE STRUCTURE NAME(RRSSTRUCT1)
           LOGSNUM(9)


    DEFINE LOGSTREAM NAME(ATR.WITPLEX.MAIN.UR)
           STRUCTNAME(RRSSTRUCT1)
           STG_DUPLEX(YES)
           DUPLEXMODE(UNCOND)
           LS_DATACLAS(SYSPLEX)
```

```
                LS_STORCLAS(LOGGER)
                HLQ(IXGLOGR)
                AUTODELETE(YES)
                RETPD(3)
         DEFINE LOGSTREAM NAME(ATR.WITPLEX.RESTART)
                STRUCTNAME(RRSSTRUCT1)
                STG_DUPLEX(YES)
                DUPLEXMODE(UNCOND)
                LS_DATACLAS(SYSPLEX)
                LS_STORCLAS(LOGGER)
                HLQ(IXGLOGR)
                AUTODELETE(YES)
                RETPD(3)
         DEFINE LOGSTREAM NAME(ATR.WITPLEX.RM.DATA)
                STRUCTNAME(RRSSTRUCT1)
                STG_DUPLEX(YES)
                DUPLEXMODE(UNCOND)
                LS_DATACLAS(SYSPLEX)
                LS_STORCLAS(LOGGER)
                HLQ(IXGLOGR)
                AUTODELETE(YES)
                RETPD(3)
         DEFINE LOGSTREAM NAME(ATR.WITPLEX.DELAYED.UR)
                STRUCTNAME(RRSSTRUCT1)
                STG_DUPLEX(YES)
                DUPLEXMODE(UNCOND)
                LS_DATACLAS(SYSPLEX)
                LS_STORCLAS(LOGGER)
                HLQ(IXGLOGR)
                AUTODELETE(YES)
                RETPD(3)
     /*

     //* DEFINE LOGSTREAM NAME(ATR.WITPLEX.ARCHIVE)
                //* STRUCTNAME(RRSSTRUCT1)
                //* STG_DUPLEX(YES)
                //* DUPLEXMODE(UNCOND)
                //* LS_DATACLAS(SYSPLEX)
                //* LS_STORCLAS(LOGGER)
                //* HLQ(IXGLOGR)
                //* AUTODELETE(YES)
                //* RETPD(3)
```

LE tuning tips for z/OS

- For best performance, use the LPALSTxx parmlib member to ensure that LE and C++ runtimes are loaded into LPA, as shown in the following example:

  Example: sys1.parmlib(LPALSTxx):

```
********************************* Top of Data ********************
USER.LPALIB,
ISF.SISFLPA,                           SDSF
CEE.SCEELPA,
                            LANGUAGE ENVIRONMENT
CBC.SCLBDLL,                           C++ RUNTIME
.
.
.

******************************** Bottom of Data ******************
```

- 5.1 + Ensure that the Language Environment data sets, SCEERUN and SCEERUN2, are authorized to enable xplink.

  Processes that run the client ORB, since they start the JVM, must run with xplink(on). For best performance in the xplink runtime, compile applications that use JNI services with xplink enabled. Enabling xplink in the runtime environment and compiling applications with xplink enabled improves performance in z/OS V1R2. As you move from z/OS V1R2 to z/OS V2R6 you should experience additional performance improvements when all the LE services WebSphere calls are xplink enabled. Not all native code in WebSphere Application Server for z/OS or companion products such as DB2 or MQ are xplink enabled and this is an area of active investigation.

- Ensure that you are **NOT** using the following options during production:
  - RPTSTG(ON)
  - RPTOPTS(ON)
  - HEAPCHK(ON)
- Turn LE heappools on.

  If you are running a client on z/OS, setting the following: SET LEPARM=′HEAPP(ON)′ in a shell script, turns on LE heappools, which should improve the performance of the client.
- Refer to “Fine tuning the LE heap.”

Fine tuning the LE heap:

The LE Heap is an area of storage management to be concerned with. For servers, IBM has compiled default values for HEAP and HEAPPOOL into the server main programs. These are good starting points for simple applications. To fine tune the LE Heap settings, use the following procedure:

1. Generate a report on storage utilization for your application servers. Use the LE function RPTSTG(ON) in a SET LEPARM= statement in JCL as shown in the example:.

   ```
   SET LEPARM='RPTOPTS(ON),RPTSTG(ON)'
   ```

   Results appear in servant joblog.
2. To bring the server down cleanly, use the following VARY command:

   ```
   VARY WLM,APPLENV=xxxx,QUIESCE
   ```

   The following example shows the servant SYSPRINT DD output from the `RPTSTG(ON)` option.

   Example:

   ```
   .   .   .
   0    HEAP statistics:
          Initial size:                                 83886080

          Increment size:                                5242880
          Total heap storage used (sugg. initial size):  184809328

          Successful Get Heap requests:                   426551
          Successful Free Heap requests:                  424262
          Number of segments allocated:                        1
          Number of segments freed:                            0
      .   .   .

     Suggested Percentages for current Cell Sizes:
       HEAPP(ON,8,6,16,4,80,42,808,45,960,5,2048,20)
     Suggested Cell Sizes:
       HEAPP(ON,32,,80,,192,,520,,1232,,2048,)

    . . .
   ```
3. Take the heap values from the ″Suggested Cell Sizes″ line in the storage utilization report and use them in another RPTSTG(ON) function to get another report on storage utilization as shown below:

```
SET LEPARM='RPTOPTS(ON),RPTSTG(ON,32,,80,,192,,520,,1232,,2048,)'
```

The following example shows the servant joblog output from the RPTOPTS(ON),RPTSTG(ON,32,,80,,192,,520,,1232,,2048,) option.

Example:

```
   .   .
0    HEAP statistics:
       Initial size:                                  83886080

       Increment size:                                 5242880
       Total heap storage used (sugg. initial size):  195803218

       Successful Get Heap requests:                    426551
       Successful Free Heap requests:                   424262
       Number of segments allocated:                         1
       Number of segments freed:                             0
   .   .   .

  Suggested Percentages for current Cell Sizes:
    HEAPP(ON,32,8,80,43,192,48,520,20,1232,5,2048,20)

  Suggested Cell Sizes:
    HEAPP(ON,32,,80,,192,,520,,1232,,2048,)
 . . .
```

4. Take the heap values from the ″Suggested Percentages for current Cell Sizes″ line of the second storage utilization report and use them in another RPTSTG(ON) function to get a third report on storage utilization as shown below:

   ```
   SET LEPARM='RPTOPTS(ON),RPTSTG(ON,32,8,80,43,192,48,520,20,1232,5,2048,20)'
   ```

   The following example shows the servant joblog output from the RPTOPTS(ON),RPTSTG(ON,32,8,80,43,192,48,520,20,1232,5,2048,20) option.

   Example:

   ```
      .   .
   0    HEAP statistics:
          Initial size:                                  83886080

          Increment size:                                 5242880
          Total heap storage used (sugg. initial size):  198372130

          Successful Get Heap requests:                    426551
          Successful Free Heap requests:                   424262
          Number of segments allocated:                         1
          Number of segments freed:                             0
      .   .   .

     Suggested Percentages for current Cell Sizes:
       HEAPP(ON,32,8,80,43,192,48,520,20,1232,5,2048,20)
     Suggested Cell Sizes:
       HEAPP(ON,32,,80,,192,,520,,1232,,2048,)
    . . .
   ```

5. On the third storage utilization report, look for the ″Total heap storage used (sugg. initial size):″ line and use this value for your initial LE heap setting. For example, in the report in third report example this value is 198372130.
6. Make sure that you remove RPTSTG since it does incur a small performance penalty to collect the storage use information.
7. For your client programs that run on z/OS or OS/390, we recommend that you at least specify HEAPP(ON) on the proc of your client to get the default LE heappools. LE will be providing additional pools (more than 6) and larger than

2048MB cell size in future releases of z/OS. You may be able to take advantage of these increased pools and cell sizes, if you have that service on your system.

8. If you use LE HEAPCHECK, make sure to turn it off once you have ensured that your code doesn't include any uninitialized storage. HEAPCHECK can be very expensive.

UNIX System Services (USS) tuning tips for z/OS

WebSphere Application Server for z/OS V5 no longer requires or recommends the shared file system for the configuration files, since it maintains its own mechanism for managing this data in a cluster. However, WebSphere for z/OS does require the shared files system for XA partner logs. Your application may also use the shared file system. This article provides some basic tuning information for the shared file system.

For basic z/OS UNIX System Services performance information, refer to the following web site:
http://www.ibm.com/servers/eserver/zseries/ebusiness/perform.html

- Mount the shared file system R/O.

  Special consideration needs to be made to file system access when you run in a sysplex. If you mount the file system R/W in a shared file system environment, only one system will have local access to the files. All other systems have remote access to the files which negatively affects performance. You may choose to put all of the files for WebSphere in their own mountable file system and mount it R/O to improve performance. However, to change your current application or install new applications, the file system must be mounted R/W. You will need to put operational procedures in place to ensure that the file system is mounted R/W when updating or installing applications.
- HFS files caching.

  HFS Files Caching Read/Write files are cached in kernel dataspaces. In order to determine what files would be good candidates for file caching you can use SMF 92 records.

  Initial cache size is defined in BPXPRMxx.
- Consider using zFS.

  z/OS has introduced a new file system called zFS which should provide improved file system access. You may benefit from using the zFS for your UNIX file system. See *z/OS UNIX System Services Planning* for more information.
- Use the filecache command.

  High activity, read-only files can be cached in the USS kernel using the filecache command. Access to files in the filecache can be much more efficient than access to files in the shared file system, even if the shared file system files are cached in dataspaces. GRS latch contention, which sometimes is an issue for frequently accessed files shared file system, will not affect files in the filecache.

  To filecache important files at startup, you can add filecache command to your /etc/rc file. Unfortunately, files which are modified after being added to the filecache may not be eligible for caching until the file system is unmounted and remounted, or until the system is re-IPLed. Refer to *z/OS UNIX System Services Command Reference* for more information about the filecache command.

  Example of using the filecache command:

  ```
  /usr/sbin/filecache -a /usr/lpp/WebSphere/V5R0M0/MQSeries/java/samples/base/
  de_DE/mqsample.html
  ```

Note: For space considerations, the above command is shown on two lines. The actual command should be on one line with no space between ″base/″ and ″de_DE...″

Workload management (WLM) tuning tips for z/OS

If you are running z/OS 1.2 or higher, you may choose to use the dynamic application environment. In that case, use the Administrative console to turn on the dynamic application environment. Use the Administrative console to provide the JCL proc name for the servant and the JCL Parm for the servant. Whether or not you use the dynamic application environment, you will need to follow the instructions below to set the WLM goals.

Setting the WLM goals properly can have a very significant effect on application throughput. The WebSphere for z/OS system address spaces should be given a fairly high priority. As work comes into the system, the work classification of the enclaves should be based on your business goals.

- Classify location service daemons and controllers as SYSSTC or high velocity.
- Use STC classification rules to classify velocity goals for application servers.

  Java garbage collection runs under this classification. Java GC is a CPU and storage intensive process, so if you set the velocity goal too high GC could consume more of the system resources than desired. On the other hand, if your Java heap is correctly tuned, GC for each server region should run no more than 5% of the time. Also, providing proper priority to GC processing is necessary since other work in the server region is stopped during much of the time GC is running.

  JSP compiles run under this classification. If your system is configured to do these compiles at runtime, setting the velocity goal too low could result in longer delays waiting for JSP compiles to complete.

  Application work is actually classified under the work manager.
- Application Environment for work running under servants
  - Subsystem type = CB
  - Classify based on server name, server instance name, User ID, and transaction class
  - Percentage response time goal is recommended

    It is usually a good idea to make the goals achievable. For example, a goal that 80% of the work will complete in .25 seconds is a typical goal. Velocity goals for application work are not meaningful and should be avoided.
  - Provide a high velocity default service class for CB transactions (Default is SYSOTHER)

    Your goals can be multi-period. This might be useful if you have distinctly short and long running transactions in the same service class. On the other hand, it is usually better to filter this work into a different service class if you can. Being in a different service class will place the work in a different servant which allows WLM much more latitude in managing the goals.
- Set your Application environment to ″No Limit″
  - Required if you need more than one servant per application server.
  - Under WLM, you can control how many servants can be started for each server. If you need more than one servant in a server make sure that ″No Limit″ is selected for the application environment associated with your server. For information about setting up WLM performance goals, see *z/OS MVS Planning: Workload Management*.

    Example:

```
Application-Environment  Notes  Options  Help
--------------------------------------------------------------------------
                    Modify an Application Environment
Command ===> ______________________________________________________________

Application Environment Name . : BBOASR2
Description  . . . . . . . . . . WAS.V40.WB02 Application server
Subsystem Type . . . . . . . . . CB
Procedure Name . . . . . . . . . BBOASR2S
Start Parameters . . . . . . . . IWMSSNM=&IWMSSNM
                                 __________________________________________
                                 __________________________________________

Limit on starting server address spaces for a subsystem instance:
1   1.  No limit

    2.  Single address space per system
    3.  Single address space per sysplex
```

Note: When the WLM configuration is set to no limit, you can control the maximum and minimum number of servants by specifying the product variables wlm_maximumSRCount=*x* and wlm_minimumSRCount=*y*. To specify these variables, click **Severs** > **Application servers** and go the application server page. **Caution:** If you specify wlm_maximumSRCount you must ensure that you specify a wlm_maximumSRCount value that is greater than or equal to the number of service classes you have defined for this application environment. Failure to do so can result in timeouts due to an insufficient number of server regions.

– Results reported in RMF Postprocessor workload activity report:
  - Transactions per second (not always the same as client tran rate)
  - Average response times (and distrubution of response times)
  - CPU time used
  - Percent response time associated with various delays

Tuning for subsystems

Steps involved in tuning the z/OS subsystems to optimize WebSphere performance include:

- "DB2 tuning tips for z/OS."
- "RACF tuning tips for z/OS" on page 104.
- "TCP/IP tuning tips for z/OS" on page 105.
- "MQ/JMS tuning tips for z/OS" on page 106.
- "GRS tuning tips for z/OS" on page 112.
- "Java virtual machine (JVM) tuning tips for z/OS" on page 112.
- "CICS tuning tips for z/OS" on page 113.

DB2 tuning tips for z/OS

Performance tuning for DB2 is usually critical to the overall performance of a WebSphere for z/OS application. DB2 is often the preferred datastore for a session or EJB. There are many books that cover DB2 tuning; we can't possibly provide as thorough a treatment of DB2 here as we would like. Listed here are some basic guidelines for DB2 tuning as well as some guidelines for tuning DB2 for WebSphere. For more complete information on DB2 tuning refer to the DB2 Universal Database for OS/390 and z/OS Administration Guide Document Number SC26-9931-03. The DB2 books can be accessed at the following Internet location:http://www.ibm.com/servers/eserver/zseries/zos/.

You have the choice of Java Database Connectivity (JDBC) or, with WebSphere Application Server Version 5.0.1, Structured Query Language in Java (SQLJ) support with both CMP and CMP beans when you are using the DB2 Universal JDBC driver provider with DB2 as your backend database. JDBC makes use of dynamic SQL whereas SQLJ generally is static and uses pre-prepared plans. You can also use SQLJ support with BMP beans when you are using the DB2 for z/OS Local JDBC provider (RRS) with DB2 for z/OS as your backend database. DB2 for z/OS users who wish to use SQLJ support with CMP beans must use the DB2 Universal JDBC driver provider. SQLJ requires an extra step to create and bind the plan whereas JDBC does not. SQLJ, as a general rule, is faster than JDBC. From a performance point of view, SQLJ it is generally a better performer than JDBC on z/OS DB2. Other features of SQLJ are that it's security model and it's repeatability are similar to static SQL which is often preferred by z/OS DB2 DBAs. SQLJ does require a couple of extra steps that are features of newer versions of WSAD (reference). Refer to Developing data access applications.

General DB2 tuning tips:

This discussion relates only to DB2 for z/OS JDBC Driver which is referred to as the DB2 for z/OS Legacy JDBC Driver.

- First, ensure that your DB2 logs are large enough, are allocated on the fastest volumes you have, and make sure they have optimal CI sizes.
- Next, ensure that you have tuned your bufferpools so that the most often-read data is in memory as much as possible.
- You many want to consider pre-formatting tables that are going to be heavily used. This avoids formatting at runtime.

DB2 for WebSphere tuning tips:

- Ensuring DB2 Tracing Under the DB2 for z/OS Universal Driver is Turned Off.
  - If the db2.jcc.propertiesFile jvm property has been defined to specify a DB2 jcc properties file to the WebSphere Application Server for z/OS, ensure that the following trace statements in the file are commented out if they are specified:
    ```
    # jcc.override.traceFile=<file name>
    db2.jcc.t2zosTraceFile=<file name>
    ```
  - If any of the DB2 Universal JDBC Driver datasources your applications are using are defined with a nonzero traceLevel custom property, use the WebSphere Application Server for z/OS Administrative console to set the traceLevel to zero.
- We recommend that you ensure indexes are defined on all your object primary keys. Failure to do so will result in costly tablespace scans.
- Ensure that, once your tables are sufficiently populated, you do a re-org to compact the tables. Executing RUNSTATS will ensure that the DB2 catalog statistics about table and column sizes and accesses are most current so that the best access patterns are chosen by the optimizer.
- You will have to define more connections called threads in DB2. WebSphere for z/OS uses a lot of threads. Sometimes this is the source of throughput bottlenecks since the server will wait at the create thread until one is available.
- Make sure you are current with JDBC maintenance. Many performance improvements have been made to JDBC. To determine the JDBC maintenance level, enter the following from the shell:
  ```
  java  com.ibm.db2.jcc.DB2Jcc  -version
  ```

If this returns a class not found, either you are at a level of the driver that is older and doesn't support this command or you have not issued the command properly.

- We recommend that you enable dynamic statement caching in DB2. To do this, modify your ZPARMS to say `CACHEDYN(YES) MAXKEEPD(16K)`. Depending on the application, this can make a very significant improvement in DB2 performance. Specifically, it can help JDBC and LDAP query.
- Increase DB2 checkpoint interval settings to a large value. To do this, modify your ZPARMS to include `CHKFREQ=xxxxx`, where xxxxx is set at a high value when doing benchmarks. On production systems there are other valid reasons to keep checkpoint frequencies lower, however.

Example: This example identifies zparm values discussed in this article.

```
//DB2INSTE   JOB MSGCLASS=H,CLASS=A,NOTIFY=IBMUSER
/*JOBPARM SYSAFF=*
//*****************************************************************
//* JOB NAME = DSNTIJUZ
//*
//* DESCRIPTIVE NAME = INSTALLATION JOB STREAM
//*
//*    LICENSED MATERIALS - PROPERTY OF IBM
//*    5675-DB2
//*    (C) COPYRIGHT 1982, 2000 IBM CORP.  ALL RIGHTS RESERVED.
//*
//*    STATUS = VERSION 7
//*
//* FUNCTION = DSNZPARM AND DSNHDECP UPDATES
//*
//* PSEUDOCODE =
//*   DSNTIZA  STEP  ASSEMBLE DSN6.... MACROS, CREATE DSNZPARM
//*   DSNTIZL  STEP  LINK EDIT DSNZPARM
//*   DSNTLOG  STEP  UPDATE PASSWORDS
//*   DSNTIZP  STEP  ASSEMBLE DSNHDECP DATA-ONLY LOAD MODULE
//*   DSNTIZQ  STEP  LINK EDIT DSNHDECP LOAD MODULE
//*   DSNTIMQ  STEP  SMP/E PROCESSING FOR DSNHDECP
//*
//* NOTES = STEP DSNTIMQ MUST BE CUSTOMIZED FOR SMP.  SEE THE NOTES
//*         NOTES PRECEDING STEP DSNTIMQ BEFORE RUNNING THIS JOB.
//*
//*  LOGLOAD=16000000,
//*******************************************************************/
//*
//DSNTIZA EXEC PGM=ASMA90,PARM='OBJECT,NODECK'
//STEPLIB DD DSN=ASM.SASMMOD1,DISP=SHR
//SYSLIB   DD  DISP=SHR,
//         DSN=DB2710.SDSNMACS
//         DD  DISP=SHR,
//         DSN=SYS1.MACLIB
//SYSLIN   DD  DSN=&LOADSET(DSNTILMP),DISP=(NEW,PASS),
//             UNIT=SYSALLDA,
//             SPACE=(800,(50,50,2)),DCB=(BLKSIZE=800)
//SYSPRINT DD  SYSOUT=*
//SYSUDUMP DD  SYSOUT=*
//SYSUT1   DD  UNIT=SYSALLDA,SPACE=(800,(50,50),,,ROUND)
//SYSUT2   DD  UNIT=SYSALLDA,SPACE=(800,(50,50),,,ROUND)
//SYSUT3   DD  UNIT=SYSALLDA,SPACE=(800,(50,50),,,ROUND)
//SYSIN    DD  *
    DSN6ENV    MVS=XA
    DSN6SPRM   RESTART,                X
               .
               .
               .
               AUTH=YES,
                                              X
```

```
            AUTHCACH=1024,                                         X
            BINDNV=BINDADD,                                        X
            BMPTOUT=4,                                             X
            CACHEDYN=YES,
                                        X
            .
            .
            .
            MAXKEEPD=16000,
                                      X
            .
            .
            .
DSN6ARVP    ALCUNIT=CYL,                                           X
            .
            .
            .
DSN6LOGP    DEALLCT=(0),                                           X
            .
            .
            .
DSN6SYSP    AUDITST=NO,                                            X
            BACKODUR=5,                                            X
           CHKFREQ=16000000,
                                    X
            CONDBAT=400,                                           X
            CTHREAD=1200,                                          X
            DBPROTCL=PRIVATE,                                      X
            DLDFREQ=5,                                             X
            DSSTIME=5,                                             X
            EXTRAREQ=100,                                          X
            EXTRASRV=100,                                          X
            EXTSEC=NO,                                             X
            IDBACK=1800,
                                          X
            .
            .
            .
          //*
```

RACF tuning tips for z/OS

Follow these guidelines for RACF tuning:

- As is always the case, don't turn things on unless you need them. In general, the cost of security has been highly optimized. However, if you don't need EJBROLEs, then don't enable the class in RACF.
- Use the RACLIST to place into memory those items that will improve performance. Specifically, ensure that you RACLIST (if used):
  - CBIND
  - EJBROLE
  - SERVER
  - STARTED

  Example:

  ```
  RACLIST (CBIND, EJBROLE, SERVER, STARTED)
  ```
- Use of things like SSL come at a price. If you are a heavy SSL user, ensure that you have appropriate hardware, such as PCI crypto cards, to speed up the handshake process.
- Here's how you define the BPX.SAFFASTPATH facility class profile. This profile allows you to bypass SAF calls which can be used to audit successful shared file system accesses.
  - Define the facility class profile to RACF.

    ```
    RDEFINE FACILITY BPX.SAFFASTPATH UACC(NONE)
    ```

– Activate this change by doing one of the following:
  - re-IPL
  - invoke the SETOMVS or SET OMVS operator commands.

Note: Do not use this option if you need to audit successful HFS accesses or if you use the IRRSXT00 exit to control HFS access.

- Use VLF caching of the UIDs and GIDs as shown in the example COFVLFxx parmlib member below:

  Example: sys1.parmlib(COFVLFxx):

```
********************************* Top of Data ********************.
.

CLASS NAME(IRRGMAP) EMAJ(GMAP)
CLASS NAME(IRRUMAP) EMAJ(UMAP)
CLASS NAME(IRRGTS) EMAJ(GTS)
CLASS NAME(IRRACEE) EMAJ(ACEE)
.
******************************** Bottom of Data ******************
```

  To avoid a costly scan of the RACF databases, make sure all HFS files have valid GIDs and UIDs.

- 5.1 + If you are using z/OS R10 or later, use the Application Identity Mapping (AIM) stage 3 support, which provides a more consistent way of mapping UIDs and GIDs to MVS users and groups. IRRGMAP and IRRUMAP are only needed if AIM has not been fully implemented. For more information, refer to the ″Performance Considerations″ section of the RACF Systems Programmer’s Guide.

TCP/IP tuning tips for z/OS

TCP/IP can be the source of some significant remote method delays. Follow these tips to tune TCP/IP:

- First, ensure that you have defined enough sockets to your system and that the default socket time-out of 180 seconds is not too high. To allow enough sockets, update the BPXPRMxx parmlib member:
  – Set MAXSOCKETS for the AF_INET filesystem high enough.
  – Set MAXFILEPROC high enough.

    We recommend setting MAXSOCKETS and MAXFILEPROC to at least 5000 for low-throughput, 10000 for medium-throughput, and 35000 for high-throughput WebSphere transaction environments. Setting high values for these parameters should not cause excessive use of resources unless the sockets or files are actually allocated.

  Example:

```
/* Open/MVS Parmlib Member                                        */
/* CHANGE HISTORY:                                                */
/*   01/31/02 AEK Increased MAXSOCKETS on AF_UNIX from 10000 to 50000*/
/*                per request from Michael Everett                */
/*   10/02/01 JAB Set up shared HFS                               */

/* KERNEL RESOURCES             DEFAULT            MIN MAX        */
/* ========================      ================== === =========== */
   .
   .
   MAXFILEPROC(65535)           /* 64               3   65535      */

   .
   .
    NETWORK DOMAINNAME(AF_INET) DOMAINNUMBER(2) MAXSOCKETS(30000)
   .
```

- Next check the specification of the port in TCPIP profile dataset to ensure that NODELAYACKS is specified as follows:

  ```
  PORT 8082 TCP NODELAYACKS
  ```

  In your runs, changing this could improve throughput by as much as 50% (this is particularly useful when dealing with trivial workloads). This setting is important for good performance when running SSL.
- You should ensure that your DNS configuration is optimized so that lookups for frequently-used servers and clients are being cached.

  Caching is sometimes related to the name server's Time To Live (TTL) value. On the one hand, setting the TTL high will ensure good cache hits. However, setting it high also means that, if the Daemon goes down, it will take a while for everyone in the network to be aware of it.

  A good way to verify that your DNS configuration is optimized is to issue the oping and onslookup USS commands. Make sure they respond in a reasonable amount of time. Often a misconfigured DNS or DNS server name will cause delays of 10 seconds or more.
- Increase the size of the TCPIP send and receive buffers from the default of 16K to at least 64K. This is the size of the buffers including control information beyond what is present in the data that you are sending in your application.To do this specify the following:

  ```
  TCPCONFIG TCPSENDBFRSIZE 65535
            TCPRCVBUFRSIZE 65535
  ```

  Note: It would not be unreasonable, in some cases, to specify 256K buffers.
- Increase the default listen backlog. This is used to buffer spikes in new connections which come with a protocol like HTTP. The default listen backlog is 10 requests. We recommend that you increase this value to something larger. For example:

  ```
  protocol_http_backlog=100
  protocol_https_backlog=100
  protocol_iiop_backlog=100
  protocol_ssl_backlog=100
  ```
- Reduce the finwait2 time.

  In the most demanding benchmarks you may find that even defining 65K sockets and file descriptors does not give you enough 'free' sockets to run 100%. When a socket is closed abnormally (for example, no longer needed) it is not made available immediately. Instead it is placed into a state called finwait2 (this is what shows up in the netstat -s command). It waits there for a period of time before it is made available in the free pool. The default for this is 600 seconds.

  Note: Unless you have trouble using up sockets, we recommend that you leave this set to the default value.

  If you are using z/OS V1.2 or above, you can control the amount of time the socket stays in finwait2 state by specifying the following in the configuration file:

  ```
  FINWAIT2TIME 60
  ```

MQ/JMS tuning tips for z/OS

JMS tuning tips

WebSphere Application Server Version 5 supports 3 different JMS providers:
- Integral JMS provider (IJP)
- MQ series JMS provider
- Generic JMS - used when a third-party JMS provider is used.

Tuning JMS involves the use of the JMS provider, design requirements of the applications, and the use of message driven beans (MDB). The JMS provider could be any of the three listed above. Application design depends on the requirements of the applications, which fall under two main categories:

- Quality of service
  - Guaranteed delivery
  - Persistence
  - Transaction
  - Security
- Messaging programming style
  - Point to point (P2P)
  - Publish/Subscribe

When using IJP as the JMS provider, we highly recommend that you not change any settings, since WebSphere has a view of what it thinks is going on. The following tuning tips are applicable in case of both IJP and MQ series JMS

Non-persistent, non-durable and transaction-not-supported messages perform better, but at the cost that they are non-recoverable due to the lack of persistence.

- Turn off all tracing. Use the display trace command to get the trace number.

  ```
  D TRACE
  ```

  Then use the stop trace global command to turn off the specific trace number

  ```
  STOP TRACE (G) TNO(xxx)
  ```

- Create long-lived queue manager connections rather than creating and destroying connections on every message.

  Reuse connections and sessions.
- Use bindings mode if your queue manager and client both reside on the same zOS image. Bindings mode is a cross-memory interface that eliminates the need for MQ to call TCP/IP. It can be implemented by not setting the TransportType on the connection. Use bindings only in case of MQ series JMS provider
- Client acknowledgements required that an ack be received from the client before another message is sent. Auto acknowledgements are a better choice and reduce delays.
- Performance will be best if the client and queue manager have the same CCSID so the queue manager does not need to translate message headers.
- Small messages are best. Use of system resources and throughput will be proportional to the size of MQ messages. However, if you must send large amounts of data, one larger message is preferable to multiple small ones. If using very large messages (for example, over 1MB) see ″WebSphere MQ Tips″.
- Persistent, transacted messages perform better than persisted non-transacted messages because multiple MQ commits can be delayed until the end of the transaction.
- Express (nonpersistent) messages perform most optimally, so use them if your application does not require persistence.
- MQ Application Server Framework (ASF) generally adds more overhead then non-ASF messaging.
- MQ local queues defined as DEFSOPT(SHARED) with the SHARE option, and shared by multiple threads or processes generally perform better than non-shared queues and use fewer resources.

- MDBs are asynchronous by nature, therefore, should never be forced to run in a serial mode (maxsession=1). Set a realistic number for maxsession for the number of concurrent sessions.
- If the JMS server fails to start with the following exception: `BBOO0220E MSGS0001E: Starting the JMSServer failed with the exception: com.ibm.ws390.messaging.QueueManagerException:Timeout processing command`, perform these steps:
  1. Add the custom property *control_region_jms_request_timeout* under the additional properties for the application server.
  2. Set the custom property to an appropriate value that allows successful initialization of the JMS Server. Value may vary per installation since it depends on each system performance. The default setting is 60 seconds and the range is 0 to 2147040.
  3. Restart the server.

MQ tuning tips

- Turn off MQ tracing in the MQ ZPARMS by specifying:.

```
TRACSTR=NO,          TRACING AUTO START                X
```

Example:

```
//*
//*             Assemble step for CSQ6LOGP
//*
//LOGP   EXEC PGM=ASMA90,PARM='DECK,NOOBJECT,LIST,XREF(SHORT)',
//             REGION=4M

//SYSLIB   DD DSN=MQSERIES.V5R3M0.SCSQMACS,DISP=SHR


//         DD DSN=SYS1.MACLIB,DISP=SHR
//SYSUT1   DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSPUNCH DD DSN=&&LOGP;,
//            UNIT=SYSDA,DISP=(,PASS),
//            SPACE=(400,(100,100,1))
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
         CSQ6LOGP INBUFF=60,         LOG INPUT BUFFER SIZE (KB)        X
               MAXRTU=2,             MAX ALLOCATED ARCHIVE LOG UNITS   X
               DEALLCT=0,            ARCHIVE LOG DEALLOCATE INTERVAL   X
               MAXARCH=500,          MAX ARCHIVE LOG VOLUMES           X
               OFFLOAD=YES,          ARCHIVING ACTIVE                  X
               OUTBUFF=4000,         LOG OUTPUT BUFFER SIZE (KB)       X
               TWOACTV=YES,          DUAL ACTIVE LOGGING               X
               TWOARCH=YES,          DUAL ARCHIVE LOGGING              X
               TWOBSDS=YES,          DUAL BSDS                         X
               WRTHRSH=20            ACTIVE LOG BUFFERS
         END
/*
//*
//*             Assemble step for CSQ6ARVP
//*
//ARVP   EXEC PGM=ASMA90,COND=(0,NE),
//             PARM='DECK,NOOBJECT,LIST,XREF(SHORT)',
//             REGION=4M

//SYSLIB   DD DSN=MQSERIES.V5R3M0.SCSQMACS,DISP=SHR


//         DD DSN=SYS1.MACLIB,DISP=SHR
//SYSUT1   DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSPUNCH DD DSN=&&ARVP;,
//            UNIT=SYSDA,DISP=(,PASS),
//            SPACE=(400,(100,100,1))
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
         CSQ6ARVP ALCUNIT=BLK,       UNITS FOR PRIQTY/SECQTY           X
               ARCPFX1=WIT.MQ1.WITA, DSN PREFIX FOR ARCHIVE LOG 1      X
               ARCPFX2=WIT.MQ2.WITA, DSN PREFIX FOR ARCHIVE LOG 2      X
               ARCRETN=9999,         ARCHIVE LOG RETENION TIME (DAYS)  X
               ARCWRTC=(1,3,4),      ARCHIVE WTO ROUTE CODE            X
               ARCWTOR=NO,           PROMPT BEFORE ARCHIVE LOG MOUNT   X
               BLKSIZE=24576,        ARCHIVE LOG BLOCKSIZE             X
               CATALOG=YES,          CATALOG ARCHIVE LOG DATA SETS     X
               COMPACT=NO,           ARCHIVE LOGS COMPACTED            X
               PRIQTY=4320,          PRIMARY SPACE ALLOCATION          X
               PROTECT=NO,           DISCRETE SECURITY PROFILES        X
               QUIESCE=5,            MAX QUIESCE TIME (SECS)           X
               SECQTY=540,           SECONDARY SPACE ALLOCATION        X
               TSTAMP=YES,           TIMESTAMP SUFFIX IN DSN           X
               UNIT=DASD,            ARCHIVE LOG DEVICE TYPE 1         X
               UNIT2=                ARCHIVE LOG DEVICE TYPE 2
         END
/*
//*
//*             Assemble step for CSQ6SYSP
//*
//SYSP   EXEC PGM=ASMA90,COND=(0,NE),
//             PARM='DECK,NOOBJECT,LIST,XREF(SHORT)',
//             REGION=4M

//SYSLIB   DD DSN=MQSERIES.V5R3M0.SCSQMACS,DISP=SHR


//         DD DSN=SYS1.MACLIB,DISP=SHR
//SYSUT1   DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSPUNCH DD DSN=&&SYSP;,
//            UNIT=SYSDA,DISP=(,PASS),
//            SPACE=(400,(100,100,1))
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *

CSQ6SYSP CTHREAD=600,       TOTAL NUMBER OF CONNECTIONS
 X
               CMDUSER=CSQOPR,       DEFAULT USERID FOR COMMANDS       X
               EXITLIM=30,           EXIT TIMEOUT (SEC)                X
               EXITTCB=8,            NUMBER OF EXIT SERVER TCBS        X

IDBACK=500,            NUMBER OF NON-TSO CONNECTIONS
X
               IDFORE=100,           NUMBER OF TSO CONNECTIONS         X

LOGLOAD=900000,       LOG RECORD CHECKPOINT NUMBER
X
               OTMACON=(,,DFSYDRU0,2147483647,CSQ),  OTMA PARAMETERS   X
               QMCCSID=0,            QMGR CCSID                        X
               QSGDATA=(,,,),        QUEUE-SHARING GROUP DATA          X
               RESAUDIT=YES,         RESLEVEL AUDITING                 X
               ROUTCDE=1,            DEFAULT WTO ROUTE CODE            X
               SMFACCT=NO,           GATHER SMF ACCOUNTING             X
               SMFSTAT=NO,           GATHER SMF STATS                  X
               STATIME=30,           STATISTICS RECORD INTERVAL (MIN)  X
               TRACSTR=NO,           TRACING AUTO START                X

               TRACTBL=99,           GLOBAL TRACE TABLE SIZE X4K       X
               WLMTIME=30,           WLM QUEUE SCAN INTERVAL (SEC)     X
               SERVICE=0             IBM SERVICE USE ONLY
         END
/*
//*
```

```
//*  LINKEDIT CSQARVP, CSQLOGP and CSQSYSP into a
//*  system parameter module.
//*
//LKED   EXEC PGM=IEWL,COND=(0,NE),
//      PARM='SIZE=(900K,124K),RENT,NCAL,LIST,AMODE=31,RMODE=ANY'
//*
//*   OUPUT AUTHORIZED APF LIBRARY FOR THE NEW SYSTEM
//*   PARAMETER MODULE.
//*
//SYSLMOD  DD DSN=SYS1.WITA.LINKLIB,DISP=SHR
//SYSUT1   DD UNIT=SYSDA,DCB=BLKSIZE=1024,
//            SPACE=(1024,(200,20))
//SYSPRINT DD SYSOUT=*
//ARVP     DD DSN=&&ARVP;,DISP=(OLD,DELETE)
//LOGP     DD DSN=&&LOGP;,DISP=(OLD,DELETE)
//SYSP     DD DSN=&&SYSP;,DISP=(OLD,DELETE)
//*
//*   LOAD LIBRARY containing the default system
//*   parameter module (CSQZPARM).
//*
//OLDLOAD  DD DSN=MQSERIES.V520.SCSQAUTH,DISP=SHR
//SYSLIN   DD *
   INCLUDE SYSP
   INCLUDE ARVP
   INCLUDE LOGP
   INCLUDE OLDLOAD(CSQZPARM)
 ENTRY CSQZMSTR
 NAME CSQZPARM(R)                  Your system parameter module name
/*
```

- Similarly, you are advised to turn off tracing in the channel initiator. Unlike base MQ tracing, this parameter cannot be enabled dynamically. To turn tracing back on for debug purposes, you will need to reassemble your MQ XPARMS. Enabling channel initiator tracing can degrade your system by 5-10%.

  The variable to set is as follows:

  ```
  TRAXSTR=NO,       START TRACE AUTOMATICALLY  YES|NO
  ```

- For improved performance in the laboratory, the following WebSphere MQ tuning parameters required modification from their default settings.The difference in most cases was significant.
  - Optimize the number of concurrent queue manager connections

    ```
    CTHREAD parameter of CSQ6SYSP (Maximum number of  concurrent
                                   connections to MQ)
    IDBACK parameter of CSQ6SYSP (Maximum number of background concurrent
                                  threads connected to MQ)
    ```

    CTHREAD is the maximum number of simultaneous connections to the queue manager. It should be greater than, or equal to, the sum of IDFORE and IDBACK. IDFORE is the number of concurrent TSO connections to WebSphere MQ, and IDBACK is the number of concurrent background connections, which includes jms threads. If any or all of these parameters are too low, applications will be unable to connect to the queue manager. These parameters are found in the CSQ6SYSP section of the MQ ZPARMS. You can update the MQ ZPARMS at any time; the updates will take effect the next time you restart the queue manager.
  - Adjust the WebSphere MQ checkpoint interval and active log buffers

    ```
    LOGLOAD parameter of CSQ6SYSP (number of log records written
                                   before a checkpoint)
    WRTHRSH parameter of CSQ6LOGP (Number of active log buffers)
    ```

    When using persistent messages, it is particularly important to pay attention to logging characteristics. MQ logs should always be placed on high performance volumes with DASD fast write enabled. MQ logs are often the single most significant bottleneck when using persistent messages. The

LOGLOAD parameter controls the number of log records written before a checkpoint (assuming, of course, that the log is large enough to hold this number of records). Checkpoints generally should occur no more frequently than every 5 or 6 minutes. If your MQ checkpoints are more frequent, you may need to increase either the size of the logs, the value of LOGLOAD, or both. In the laboratory, we use a LOGLOAD value of 900000 because we execute high throughput jms workloads that are very write-intensive. When the space on the log is exhausted, a log switch occurs which disrupts performance. You can avoid frequent log switches by increasing the size of the log(s).

WRTHRSH is the number of active log buffers, and determines how much data is held in memory before a log write occurs. If you have a high I/O rate to your log volume(s), you may wish to increase this parameter. In the laboratory, we use a value of 200.

Generally speaking, your message rate for persistent messages cannot exceed the bandwidth capacity of your slowest log volume. For example, if your were sending messages of 5KB at a throughput rate of 300 per second, you would be writing at least 1.9 MB of data per second to the log (this is roughly 1.3KB plus the user message size for each logged message).

– Specify the size of the archive logs

```
SECQTY parameter of CSQ6ARVP  (archive log space allocation)
PRIQTY parameter of CSQ6ARVP  (archive log space allocation)
ALCUNIT parameter of CSQ6ARVP  (archive Log allocation unit)
TWOARCH parameter of CSQ6LOGP  (dual archive logs)
```

The PRIQTY and SECQTY parameters control the size of the archive logs. Generally it is best to allocate them (ALCUNIT) in cylinders in lieu of blocks. Depending upon your data integrity requirements, you may or may not choose to have dual archive logging.

– Specify the number of buffers

```
BUFFERS  (number of buffers on the DEFINE BUFFERPOOL statement)
```

Use the DEFINE BUFFERPOOL statement to specify the number of buffers. It is important to insure the number of buffers is large enough to hold at least an entire message (and its headers). Otherwise, WebSphere MQ will be forced to write to the Page Data sets for every message. For example, a 100MB message requires at least 26000 pages in the bufferpool. MQ Buffer Manager statistics can be used to determine the number of times a buffer was unavailable. See Support Pac MP1B, MQSeries for OS/390 V5.2 - Interpreting accounting and statistics data.

– Specify queue definitions

```
INDXTYPE(NONE)  (index specification for queue definitions) or
INDXTYPE(CORRELID)
DEFPSIST(NO)
```

Unless your applications retrieve messages by other than correlation ID (which is the case for jms publish/subscribe) or message ID, it is normally best not to specify message selectors on queue definitions. You should, however, make sure you have specified INDXTYPE(CORRELID) on the SYSTEM.JMS.ND/D queues, and/or on the SYSTEM.JMS.ND.CC / .D.CC queues, and/or on any shared message queues for publish/subscribe. Specify DEFPSIST(NO) unless you want messages on a particular queue to default to persistent. Since DEFPSIST(YES) will affect performance, make sure you really want persistent messages.

If your applications use message ID to access the message, we highly recommend indexing the queue on message ID. You can add the index either at the time of queue definition or on the fly by using the MQSeries command INDXTYPE(MSGID).

– Specify channel definitions

```
BATCHSZ parameter for queue definitions (Number of messages
                                         sent as a batch)
```

We did not modify the BATCHSZ parameter in the laboratory.

GRS tuning tips for z/OS

WebSphere for z/OS uses GRS to communicate information between servers in a sysplex. When there are multiple servers defined in a system or a sysplex, a request may end up on the wrong server. WebSphere uses GRS to determine where the transaction is running. Therefore, if you are using global transactions, WebSphere will issue an enqueue for that transaction at the start of the transaction and hold on to that enqueue until the transaction ends. WebSphere for z/OS uses GRS enqueues for the following:

- Two-phase commit transactions involving more than one server
- HTTP sessions in memory
- Stateful EJBs
- ″Sticky″ transactions to keep track of pseudo-conversational states.
- If you **are not** in a sysplex, you should configure GRS=NONE.
- If you **are** in a sysplex, we strongly recommend GRS=STAR.

This requires configuring GRS to use the coupling facility. All of the WebSphere enqueues are issued with RNL=NO, which prevents misconfiguring the GRSRNLxx with inappropriate values. See the GRS documentation for details on setting this up.

Java virtual machine (JVM) tuning tips for z/OS

Before you begin:

1. Starting in WebSphere Version 5.1 the Java Development Kit (JDK) level required is Version 1.4.1. WebSphere Version 5.1 cannot run with JDK Version 1.3.1. JDK Version 1.4.1 is faster than JDK Version 1.3.1. Ensure that you have the most recent version of JVM that is supported by WebSphere for z/OS. As of this writing, the JVM level for WebSphere Application Server for z/OS V5 is 1.3.1 and PTF 20.
2. Have the most recent PTFs, since almost every PTF level has improved performance of the JVM.
3. Have sufficient JVM Heap Size. Refer to “Java virtual machine storage tuning tips for z/OS” on page 92 for a discussion of this setting.

How to view or set: Use the WebSphere Administrative console :

1. Click **Servers > Application Servers >** *server_name* **> Process Definition > Java Virtual Machine**
2. Select the options listed in the section belong on the Configuration Tab.

- Run with the JIT (Just In Time) compiler active.

  In the General Properties section of the Configuration Tab, ensure that Disable JIT is not selected. The default is JIT support enabled.
- Do not specify the debug version of the JVM `libjava_g` in your libpath. Severe performance degradation is likely when running with the debug version of the JVM.
- Have Classpath point to only the classes you need (the classes that are referenced most frequently should be located near the front of the path, if possible).

  In the General Properties section of the Configuration Tab, enter the Classpath in the text box of the Classpath option.

- Verify the Classpath as part of the Java configuration.
- To speed up JVM initialization and improve server startup time, specify the following command line arguments in the General JVM Arguments field in the General Properties section of the Configuration Tab.

  ```
  -Xquickstart
  -Xverify:none
  ```

  For JDK Version 1.4.1 this reduces servant startup time at the cost of some runtime performance. For JDK Version 1.3.1, these options can reduce servant startup time by as much as 40%. However, they will reduce runtime throughput by about 8%.
- Sometimes poor performance is caused by a missing class. The class loader will look in it's tables of already loaded classes and if the class is not found to be already loaded it will search for it. This search process can cause a high amount of I/O activity to the HFS volumes. To determine if this is the problem you can collect CTRACE records from the file system. Once you determine which class is not being found you can repair the problem by providing the class or by removing the need for it.

  Note: Please see the ″Applications″ section in the WebSphere Application Server for z/OS V5.0 information center, access to which can be obtained through the WebSphere for z/OS library Web site http://www.ibm.com/software/webservers/appserv/zos_os390/library.html for more information on ″Application client troubleshooting tips.″
- For more information about JVM performance on z/OS and OS/390, see http://www.s390.ibm.com/java/perform.html.

CICS tuning tips for z/OS

These recommendations only apply to WebSphere applications that access CICS.

The LGDFINT system initialization parameter specifies the log defer interval used by CICS log manager when determining how long to delay a forced journal write request before invoking the MVS system logger. The value is specified in milliseconds. Performance evaluations of typical CICS transaction workloads have shown that the default setting of 5 milliseconds gives the best balance between response time and central processor cost. Be aware that CICS performance can be adversely affected by a change to the log defer interval value. Too high a value will delay CICS transaction throughput due to the additional wait before invoking the MVS system logger. An example of a scenario where a reduction in the log defer interval might be beneficial to CICS transaction throughout would be where many forced log writes are being issued, and little concurrent task activity is occurring. Such tasks will spend considerable amounts of their elapsed time waiting for the log defer period to expire. In such a situation, there is limited advantage in delaying a call to the MVS system logger to write out a log buffer, since few other log records will be added to the buffer during the delay period.

- Set the LGDFINT system initialization parameter to 5.

  While CICS is running, you can use the CEMT SET SYSTEM[LOGDEFER(*value*)] command to alter the LGDFINT setting dynamically.
- Set the CICS RECEIVECOUNT value high enough to handle all concurrent EXCI pipes on the system.

  The default value is 4. You set this value in the EXCI sessions resource definition.

For more detailed information on CICS, refer to the *CICS Performance Guide*.

Tuning the WebSphere Application Server for z/OS runtime

Steps involved in tuning the WebSphere Application Server for z/OS runtime to optimize performance include reviewing the:
- "Review the WebSphere for z/OS configuration"
- "Internal tracing tips for WebSphere for z/OS" on page 120
- "Location of executable programs tips for z/OS" on page 121
- "Security tuning tips for z/OS" on page 121
- 5.1+ Web server plug-in tuning tips
- "Servlet and EJB integrated runtime tuning tips for z/OS" on page 122
- "WebSphere Application Server tuning tips for use with DB2" on page 123.

Review the WebSphere for z/OS configuration

The first thing to do is review the WebSphere for z/OS configuration. One simple way to do this is to look in your application control and server regions in SDSF. When each server starts, the runtime prints out the current configuration data in the joblog. An example joblog is shown here.

```
1                    J E S 2  J O B  L O G  --  S Y S T E M  S P 8   --  N O D E  P K S T 4 0
0
 15.51.44 STC07693 ---- MONDAY,    19 APR 2004 ----
 15.51.44 STC07693  $HASP373 BBOASR4S STARTED
 15.51.44 STC07693  IEF403I BBOASR4S - STARTED - TIME=15.51.44
 15.51.44 STC07693  +BBOO0004I WEBSPHERE FOR Z/OS SERVANT PROCESS  063
    063              SP8C/SP8N/CLUSTER1/BBOASR4 IS STARTING.
 15.51.44 STC07693  +BBOO0239I WEBSPHERE FOR Z/OS SERVANT PROCESS SP8Cell/SP8Node/BBOASR4A  064
    064              IS STARTING.
 15.51.44 STC07693  +BBOM0007I CURRENT CB SERVICE LEVEL IS build level wd51g05 release date  065
    065              03/18/04 09:04:40.
 15.51.44 STC07693  +BBOM0001I               cell_name: SP8Cell.
 15.51.44 STC07693  +BBOM0001I         cell_short_name: SP8C.
 15.51.44 STC07693  +BBOM0001I client_protocol_password: NOT SET.
 15.51.44 STC07693  +BBOM0001I client_protocol_resolve_name: NOT SET.
 15.51.44 STC07693  +BBOM0001I client_protocol_resolve_port: NOT SET, DEFAULT=2809.
 15.51.44 STC07693  +BBOM0001I    client_protocol_user: NOT SET.
 15.51.44 STC07693  +BBOM0001I client_ras_logstreamname: NOT SET.
 15.51.44 STC07693  +BBOM0001I        clustered_server: 0.
 15.51.44 STC07693  +BBOM0001I com_ibm_authMechanisms_type_OID: No OID for this mechanism.
 15.51.44 STC07693  +BBOM0001I com_ibm_security_SAF_unauthenticated: CBGUEST.
 15.51.44 STC07693  +BBOM0001I com_ibm_security_SAF_EJBROLE_Audit_Messages_Suppress: NOT  076
    076              SET, DEFAULT=0.
 15.51.44 STC07693  +BBOM0001I com_ibm_userRegistries_type: security:LocalOSUserRegistry.
 15.51.44 STC07693  +BBOM0001I com_ibm_userRegistries_CustomUserRegistry_realm: NOT SET,  078
    078              DEFAULT=CustomRealm.
 15.51.44 STC07693  +BBOM0001I com_ibm_userRegistries_LDAPUserRegistry_realm: NOT SET,  079
    079              DEFAULT=LDAPRealm.
 15.51.44 STC07693  +BBOM0001I com_ibm_CSI_claim_ssl_sys_v2_timeout: NOT SET, DEFAULT=100.
 15.51.44 STC07693  +BBOM0001I com_ibm_CSI_claim_ssl_sys_v3_timeout: 600.

 15.51.44 STC07693  +BBOM0001I com_ibm_CSI_claimClientAuthenticationtype:  082
    082              SAFUSERIDPASSWORD.
 15.51.44 STC07693  +BBOM0001I com_ibm_CSI_claimClientAuthenticationRequired: 0.
 15.51.44 STC07693  +BBOM0001I com_ibm_CSI_claimClientAuthenticationSupported: 1.
 15.51.44 STC07693  +BBOM0001I com_ibm_CSI_claimIdentityAssertionSupported: 0.
 15.51.44 STC07693  +BBOM0001I com_ibm_CSI_claimIdentityAssertionTypeCert: 0.
 15.51.44 STC07693  +BBOM0001I com_ibm_CSI_claimIdentityAssertionTypeDN: 0.
 15.51.44 STC07693  +BBOM0001I com_ibm_CSI_claimIdentityAssertionTypeSAF: 0.
 15.51.44 STC07693  +BBOM0001I com_ibm_CSI_claimKeyringName: SP8Keyring.
 15.51.44 STC07693  +BBOM0001I com_ibm_CSI_claimMessageConfidentialityRequired: 0.
 15.51.44 STC07693  +BBOM0001I com_ibm_CSI_claimMessageIntegrityRequired: NOT SET,  091
    091              DEFAULT=1.
 15.51.44 STC07693  +BBOM0001I com_ibm_CSI_claimMessageIntegritySupported: NOT SET,  092
    092              DEFAULT=1.
```

```
15.51.44 STC07693  +BBOM0001I com_ibm_CSI_claimSecurityCipherSuiteList: NOT SET.
15.51.44 STC07693  +BBOM0001I com_ibm_CSI_claimSecurityLevel: HIGH.
15.51.44 STC07693  +BBOM0001I com_ibm_CSI_claimStateful: 1.
15.51.44 STC07693  +BBOM0001I com_ibm_CSI_claimTransportAssocSSLTLSRequired: 0.
15.51.44 STC07693  +BBOM0001I com_ibm_CSI_claimTransportAssocSSLTLSSupported: 1.
15.51.44 STC07693  +BBOM0001I com_ibm_CSI_claimTLClientAuthenticationRequired: 0.
15.51.44 STC07693  +BBOM0001I com_ibm_CSI_claimTLClientAuthenticationSupported: 1.
15.51.44 STC07693  +BBOM0001I com_ibm_CSI_perform_ssl_sys_v2_timeout: NOT SET,  100
   100              DEFAULT=100.
15.51.44 STC07693  +BBOM0001I com_ibm_CSI_perform_ssl_sys_v3_timeout: 600.
15.51.44 STC07693  +BBOM0001I com_ibm_CSI_performClientAuthenticationtype:  102
   102              SAFUSERIDPASSWORD.
15.51.44 STC07693  +BBOM0001I com_ibm_CSI_performClientAuthenticationRequired: 0.
15.51.44 STC07693  +BBOM0001I com_ibm_CSI_performClientAuthenticationSupported: 1.
15.51.44 STC07693  +BBOM0001I com_ibm_CSI_performIdentityAssertionRequired: 0.
15.51.44 STC07693  +BBOM0001I com_ibm_CSI_performIdentityAssertionSupported: 0.
15.51.44 STC07693  +BBOM0001I com_ibm_CSI_performKeyringName: SP8Keyring.
15.51.44 STC07693  +BBOM0001I com_ibm_CSI_performMessageConfidentialityRequired: 0.
15.51.44 STC07693  +BBOM0001I com_ibm_CSI_performMessageConfidentialitySupported: 1.
15.51.44 STC07693  +BBOM0001I com_ibm_CSI_performMessageIntegrityRequired: 1.
15.51.44 STC07693  +BBOM0001I com_ibm_CSI_performMessageIntegritySupported: 1.
15.51.44 STC07693  +BBOM0001I com_ibm_CSI_performSecurityCipherSuiteList: NOT SET.
15.51.44 STC07693  +BBOM0001I com_ibm_CSI_performSecurityLevel: HIGH.
15.51.44 STC07693  +BBOM0001I com_ibm_CSI_performStateful: 1.
15.51.44 STC07693  +BBOM0001I com_ibm_CSI_performTransportAssocSSLTLSRequired: 0.
15.51.44 STC07693  +BBOM0001I com_ibm_CSI_performTransportAssocSSLTLSSupported: 1.
15.51.44 STC07693  +BBOM0001I com_ibm_CSI_performTLClientAuthenticationRequired: 0.
15.51.44 STC07693  +BBOM0001I com_ibm_CSI_performTLClientAuthenticationSupported: 0.
15.51.44 STC07693  +BBOM0001I com_ibm_DAEMON_claim_ssl_sys_v3_timeout: NOT SET,  119
   119              DEFAULT=600.
15.51.44 STC07693  +BBOM0001I com_ibm_DAEMON_claimClientAuthentication: NOT SET,  120
   120              DEFAULT=0.
15.51.44 STC07693  +BBOM0001I com_ibm_DAEMON_claimKeyringName: NOT SET.
15.51.44 STC07693  +BBOM0001I com_ibm_DAEMON_claimSecurityCipherSuiteList: NOT SET.
15.51.44 STC07693  +BBOM0001I com_ibm_DAEMON_claimSecurityLevel: NOT SET, DEFAULT=HIGH.
15.51.44 STC07693  +BBOM0001I com_ibm_HTTP_claim_ssl_sys_v2_timeout: NOT SET, DEFAULT=100.
15.51.44 STC07693  +BBOM0001I com_ibm_HTTP_claim_ssl_sys_v3_timeout: NOT SET, DEFAULT=600.
15.51.44 STC07693  +BBOM0001I com_ibm_HTTP_claim_sslEnabled: 1.
15.51.44 STC07693  +BBOM0001I com_ibm_HTTP_claimClientAuthentication: NOT SET, DEFAULT=0.
15.51.44 STC07693  +BBOM0001I com_ibm_HTTP_claimKeyringName: NOT SET.
15.51.44 STC07693  +BBOM0001I com_ibm_HTTP_claimSecurityCipherSuiteList: NOT SET.
15.51.44 STC07693  +BBOM0001I com_ibm_HTTP_claimSecurityLevel: NOT SET, DEFAULT=HIGH.
15.51.44 STC07693  +BBOM0001I com_ibm_Server_Security_Enabled: 0.
15.51.44 STC07693  +BBOM0001I              config_root: NOT SET.
15.51.44 STC07693  +BBOM0001I control_region_classpath:  133
   133              /WebSphere/V5R1M0/AppServer/properties:/WebSphere/V5R1M0/AppServer/lib
   133              /bootstrap.jar:/WebSphere/V5R1M0/AppServer/lib/j2ee.jar:/WebSphere/V5R
   133              1M0/AppServer/lib/lmproxy.jar.
15.51.44 STC07693  +BBOM0001I control_region_jms_request_timeout: NOT SET, DEFAULT=60.
15.51.44 STC07693  +BBOM0001I control_region_jvm_localrefs: NOT SET, DEFAULT=128.
15.51.44 STC07693  +BBOM0001I control_region_jvm_logfile: NOT SET.
15.51.44 STC07693  +BBOM0001I control_region_jvm_properties_file:  137
   137              /WebSphere/V5R1M0/AppServer/config/cells/SP8Cell/nodes/SP8Node/servers
   137              /BBOASR4A/control.jvm.options.
15.51.44 STC07693  +BBOM0001I  control_region_libpath:  138
   138              /usr/lpp/java/J1.4/bin:/usr/lpp/java/J1.4/bin/classic:/WebSphere/V5R1M
   138              0/AppServer/lib.
15.51.44 STC07693  +BBOM0001I control_region_mdb_request_timeout: NOT SET, DEFAULT=120.
15.51.44 STC07693  +BBOM0001I control_region_security_enable_trusted_applications: 0.
15.51.44 STC07693  +BBOM0001I control_region_ssl_thread_pool_size: NOT SET, DEFAULT=3.
15.51.44 STC07693  +BBOM0001I control_region_thread_pool_size: 25.
15.51.44 STC07693  +BBOM0001I control_region_thread_stack_size: NOT SET, DEFAULT=0.
15.51.44 STC07693  +BBOM0001I control_region_use_java_g: NOT SET, DEFAULT=0.
15.51.44 STC07693  +BBOM0001I control_region_wlm_dispatch_timeout: 300.
15.51.44 STC07693  +BBOM0001I        daemon_group_name: SP8C.
15.51.44 STC07693  +BBOM0001I     daemon_start_command: START BBO5DMN.
15.51.44 STC07693  +BBOM0001I daemon_start_command_args:  148
```

```
   148              JOBNAME=BBODMNB,ENV=SP8C.SP8C.BBODMNB.
15.51.44 STC07693  +BBOM0001I         daemon_wlmable: NOT SET, DEFAULT=0.
15.51.44 STC07693  +BBOM0001I      daemonInstanceName: BBODMNB.
15.51.44 STC07693  +BBOM0001I              daemonName: SP8C.
15.51.44 STC07693  +BBOM0001I            nls_language: NOT SET, DEFAULT=ENUS.
15.51.44 STC07693  +BBOM0001I               node_name: SP8Node.
15.51.44 STC07693  +BBOM0001I         node_short_name: SP8N.
15.51.44 STC07693  +BBOM0001I nonauthenticated_clients_allowed: 1.
15.51.44 STC07693  +BBOM0001I protocol_accept_http_work_after_min_srs: NOT SET, DEFAULT=1.
15.51.44 STC07693  +BBOM0001I protocol_bboc_log_response_failure: NOT SET, DEFAULT=0.
15.51.44 STC07693  +BBOM0001I protocol_bboc_log_return_exception: NOT SET, DEFAULT=0.
15.51.44 STC07693  +BBOM0001I protocol_giop_level_highest: NOT SET, DEFAULT=1.2.
15.51.44 STC07693  +BBOM0001I   protocol_http_backlog: NOT SET, DEFAULT=10.
15.51.44 STC07693  +BBOM0001I   protocol_http_cloneid: NOT SET, DEFAULT=-1.
15.51.44 STC07693  +BBOM0001I protocol_http_defaultIdentity: CBGUEST.
15.51.44 STC07693  +BBOM0001I protocol_http_enableSSLTracking: 0.
15.51.44 STC07693  +BBOM0001I protocol_http_large_data_inbound_buffer: NOT SET, DEFAULT=0.
15.51.44 STC07693  +BBOM0001I protocol_http_large_data_response_buffer: NOT SET,  165
   165              DEFAULT=104857600.
15.51.44 STC07693  +BBOM0001I protocol_http_listenIPAddress: *.
15.51.44 STC07693  +BBOM0001I protocol_http_max_connect_backlog: NOT SET, DEFAULT=0.
15.51.44 STC07693  +BBOM0001I protocol_http_max_keep_alive_connections: NOT SET,  168
   168              DEFAULT=0.
15.51.44 STC07693  +BBOM0001I protocol_http_max_persist_requests: NOT SET, DEFAULT=50.
15.51.44 STC07693  +BBOM0001I      protocol_http_port: 80.
15.51.44 STC07693  +BBOM0001I protocol_http_session_cookie_name: NOT SET,  171
   171              DEFAULT=JSESSIONID.
15.51.44 STC07693  +BBOM0001I protocol_http_session_id_length: NOT SET, DEFAULT=23.
15.51.44 STC07693  +BBOM0001I protocol_http_session_no_additional_info: NOT SET,  173
   173              DEFAULT=0.
15.51.44 STC07693  +BBOM0001I protocol_http_session_persistent: 0.
15.51.44 STC07693  +BBOM0001I protocol_http_session_rewrite_identifier: NOT SET,  175
   175              DEFAULT=jsessionid.
15.51.44 STC07693  +BBOM0001I protocol_http_tcp_no_delay: NOT SET, DEFAULT=0.
15.51.44 STC07693  +BBOM0001I protocol_http_timeout_input: NOT SET, DEFAULT=120.
15.51.44 STC07693  +BBOM0001I protocol_http_timeout_output: NOT SET, DEFAULT=300.
15.51.44 STC07693  +BBOM0001I protocol_http_timeout_output_recovery: NOT SET,  179
   179              DEFAULT=SERVANT.
15.51.44 STC07693  +BBOM0001I protocol_http_timeout_persistentSession: NOT SET,  180
   180              DEFAULT=30.
15.51.44 STC07693  +BBOM0001I protocol_http_transactionClass: NOT SET.
15.51.44 STC07693  +BBOM0001I protocol_http_transport_class_mapping_file: NOT SET.
15.51.44 STC07693  +BBOM0001I protocol_http_transport_network_qos: NONE.
15.51.44 STC07693  +BBOM0001I protocol_http_trusted_proxy: NOT SET, DEFAULT=0.
15.51.44 STC07693  +BBOM0001I  protocol_https_backlog: NOT SET, DEFAULT=10.
15.51.44 STC07693  +BBOM0001I protocol_https_cert_mapping_file: NOT SET.
15.51.44 STC07693  +BBOM0001I protocol_https_default_cert_label: NOT SET.
15.51.44 STC07693  +BBOM0001I protocol_https_default_identity: CBGUEST.
15.51.44 STC07693  +BBOM0001I protocol_https_listenIPAddress: *.
15.51.44 STC07693  +BBOM0001I protocol_https_max_connect_backlog: NOT SET, DEFAULT=0.
15.51.44 STC07693  +BBOM0001I protocol_https_max_keep_alive_connections: NOT SET,  191
   191              DEFAULT=0.
15.51.44 STC07693  +BBOM0001I protocol_https_max_persist_requests: NOT SET, DEFAULT=50.
15.51.44 STC07693  +BBOM0001I protocol_https_mutual_auth_cbind_check: NOT SET, DEFAULT=0.
15.51.44 STC07693  +BBOM0001I     protocol_https_port: 443.
15.51.44 STC07693  +BBOM0001I protocol_https_tcp_no_delay: NOT SET, DEFAULT=0.
15.51.44 STC07693  +BBOM0001I protocol_https_timeout_input: NOT SET, DEFAULT=120.
15.51.44 STC07693  +BBOM0001I protocol_https_timeout_output: NOT SET, DEFAULT=300.
15.51.44 STC07693  +BBOM0001I protocol_https_timeout_output_recovery: NOT SET,  198
   198              DEFAULT=SERVANT.
15.51.44 STC07693  +BBOM0001I protocol_https_timeout_persistentSession: NOT SET,  199
   199              DEFAULT=30.
15.51.44 STC07693  +BBOM0001I protocol_https_transactionClass: NOT SET.
15.51.44 STC07693  +BBOM0001I protocol_https_transport_class_mapping_file: NOT SET.
15.51.44 STC07693  +BBOM0001I protocol_https_transport_network_qos: NOT SET.
15.51.44 STC07693  +BBOM0001I protocol_https_trusted_proxy: NOT SET, DEFAULT=0.
15.51.44 STC07693  +BBOM0001I   protocol_iiop_backlog: NOT SET, DEFAULT=10.
```

```
15.51.44 STC07693  +BBOM0001I protocol_iiop_backlog_ssl: NOT SET, DEFAULT=10.
15.51.44 STC07693  +BBOM0001I protocol_iiop_daemon_listenIPAddress:  206
   206              BOSSQ098.PDL.POK.IBM.COM.
15.51.44 STC07693  +BBOM0001I protocol_iiop_daemon_port: 5655.
15.51.44 STC07693  +BBOM0001I protocol_iiop_daemon_port_ssl: 5656.
15.51.44 STC07693  +BBOM0001I protocol_iiop_listenIPAddress: *.
15.51.44 STC07693  +BBOM0001I protocol_iiop_local_timeout: 1800.
15.51.44 STC07693  +BBOM0001I protocol_iiop_no_local_copies: 0.
15.51.44 STC07693  +BBOM0001I      protocol_iiop_port: 2809.
15.51.44 STC07693  +BBOM0001I  protocol_iiop_port_ssl: 2808.
15.51.44 STC07693  +BBOM0001I protocol_iiop_propagate_unknown_service_ctxs: NOT SET.
15.51.44 STC07693  +BBOM0001I protocol_iiop_server_session_keepalive: 0.
15.51.44 STC07693  +BBOM0001I protocol_iiop_server_session_keepalive_ssl: 0.
15.51.44 STC07693  +BBOM0001I        ras_debugEnabled: NOT SET, DEFAULT=0.
15.51.44 STC07693  +BBOM0001I      ras_default_msg_dd: NOT SET.
15.51.44 STC07693  +BBOM0001I ras_dumpoptions_dumptype: NOT SET, DEFAULT=3.
15.51.44 STC07693  +BBOM0001I ras_dumpoptions_ledumpoptons: NOT SET, DEFAULT=THREAD(ALL)  220
   220              BLOCKS.
15.51.44 STC07693  +BBOM0001I     ras_hardcopy_msg_dd: NOT SET.
15.51.44 STC07693  +BBOM0001I   ras_log_logstreamName: NOT SET.
15.51.44 STC07693  +BBOM0001I    ras_minorcode_action: NOT SET, DEFAULT=NODIAGNOSTICDATA.
15.51.44 STC07693  +BBOM0001I          ras_time_local: NOT SET, DEFAULT=0.
15.51.44 STC07693  +BBOM0001I         ras_trace_basic: NOT SET.
15.51.44 STC07693  +BBOM0001I   ras_trace_ctraceParms: 50.
15.51.44 STC07693  +BBOM0001I ras_trace_defaultTracingLevel: 1.
15.51.44 STC07693  +BBOM0001I        ras_trace_detail: NOT SET.
15.51.44 STC07693  +BBOM0001I ras_trace_exclude_specific: NOT SET.
15.51.44 STC07693  +BBOM0001I ras_trace_minorCodeTraceBacks: NOT SET.
15.51.44 STC07693  +BBOM0001I ras_trace_outputLocation: SYSPRINT BUFFER.
15.51.44 STC07693  +BBOM0001I      ras_trace_specific: NOT SET.
15.51.44 STC07693  +BBOM0001I   ras_trace_BufferCount: 4.
15.51.44 STC07693  +BBOM0001I    ras_trace_BufferSize: 128K.
15.51.44 STC07693  +BBOM0001I  read_license_agreement: 1.
15.51.44 STC07693  +BBOM0001I security_assertedID_IBM_accepted: 0.
15.51.44 STC07693  +BBOM0001I security_assertedID_IBM_sent: 0.
15.51.44 STC07693  +BBOM0001I security_kerberos_allowed: 0.
15.51.44 STC07693  +BBOM0001I security_local_identity: CBGUEST.
15.51.44 STC07693  +BBOM0001I security_remote_identity: CBGUEST.
15.51.44 STC07693  +BBOM0001I security_sslClientCerts_allowed: 0.
15.51.44 STC07693  +BBOM0001I     security_sslKeyring: NOT SET.
15.51.44 STC07693  +BBOM0001I       security_sslType1: 0.
15.51.44 STC07693  +BBOM0001I security_userid_passticket_allowed: 1.
15.51.44 STC07693  +BBOM0001I security_userid_password_allowed: 0.
15.51.44 STC07693  +BBOM0001I security_zOS_domainName: NOT SET.
15.51.44 STC07693  +BBOM0001I security_zOS_domainType: 0.
15.51.44 STC07693  +BBOM0001I security_zSAS_ssl_repertoire: SP8Node/DefaultIIOPSSL.
15.51.44 STC07693  +BBOM0001I security_EnableRunAsIdentity: 0.
15.51.44 STC07693  +BBOM0001I security_EnableSyncToOSThread: 0.
15.51.44 STC07693  +BBOM0001I server_configured_system_name: SP8.
15.51.44 STC07693  +BBOM0001I server_generic_short_name: CLUSTER1.
15.51.44 STC07693  +BBOM0001I     server_generic_uuid:  253
   253              BAFFE1A54D7D1043000000011400000010C0A83DC6.
15.51.44 STC07693  +BBOM0001I server_region_classpath:  254
   254              /WebSphere/V5R1M0/AppServer/properties:/WebSphere/V5R1M0/AppServer/lib
   254              /bootstrap.jar:/WebSphere/V5R1M0/AppServer/lib/urlprotocols.jar:/WebSp
   254              here/V5R1M0/AppServer/lib/j2ee.jar:/WebSphere/V5R1M0/AppServer/lib/lmp
   254              roxy.jar.
15.51.44 STC07693  +BBOM0001I server_region_dynapplenv_jclparms:  255
   255              JOBNAME=&IWMSSNM.S,ENV=SP8C.SP8N.&IWMSSNM    255              .
15.51.44 STC07693  +BBOM0001I server_region_dynapplenv_jclproc: BBO5ASR .
15.51.44 STC07693  +BBOM0001I server_region_jvm_localrefs: NOT SET, DEFAULT=128.
15.51.44 STC07693  +BBOM0001I server_region_jvm_logfile: NOT SET.
15.51.44 STC07693  +BBOM0001I server_region_jvm_properties_file:  259
   259              /WebSphere/V5R1M0/AppServer/config/cells/SP8Cell/nodes/SP8Node/servers
   259              /BBOASR4A/servant.jvm.options.
15.51.44 STC07693  +BBOM0001I   server_region_libpath:  260
   260              /usr/lpp/java/J1.4/bin:/usr/lpp/java/J1.4/bin/classic:/WebSphere/V5R1M
   260              0/AppServer/lib.
15.51.44 STC07693  +BBOM0001I server_region_recycle_count: NOT SET, DEFAULT=0.
15.51.44 STC07693  +BBOM0001I server_region_thread_stack_size: NOT SET, DEFAULT=0.
15.51.44 STC07693  +BBOM0001I server_region_use_java_g: NOT SET, DEFAULT=0.
15.51.44 STC07693  +BBOM0001I server_region_workload_profile: IOBOUND.
15.51.44 STC07693  +BBOM0001I    server_specific_name: BBOASR4A.
15.51.44 STC07693  +BBOM0001I server_specific_short_name: BBOASR4.
15.51.44 STC07693  +BBOM0001I    server_specific_uuid:  267
   267              BAFFE1A54C460F430000011400000010C0A83DC6.
15.51.44 STC07693  +BBOM0001I server_start_wait_for_initialization_Timeout: NOT SET,  268
   268              DEFAULT=0.
15.51.44 STC07693  +BBOM0001I             server_type: AppServer.
15.51.44 STC07693  +BBOM0001I server_SMF_container_activity_enabled: NOT SET, DEFAULT=0.
15.51.44 STC07693  +BBOM0001I server_SMF_container_interval_enabled: NOT SET, DEFAULT=0.
15.51.44 STC07693  +BBOM0001I server_SMF_interval_length: NOT SET, DEFAULT=0.
15.51.44 STC07693  +BBOM0001I server_SMF_server_activity_enabled: NOT SET, DEFAULT=0.
15.51.44 STC07693  +BBOM0001I server_SMF_server_interval_enabled: NOT SET, DEFAULT=0.
15.51.44 STC07693  +BBOM0001I          serverRegionid: NOT SET.
15.51.44 STC07693  +BBOM0001I shell_command_proc_name: BBOW5SH.
15.51.44 STC07693  +BBOM0001I transaction_defaultTimeout: 120.
15.51.44 STC07693  +BBOM0001I transaction_maximumTimeout: 300.
15.51.44 STC07693  +BBOM0001I transaction_recoveryTimeout: NOT SET, DEFAULT=15.
15.51.44 STC07693  +BBOM0001I            was_env_file:  280
   280              /WebSphere/V5R1M0/AppServer/config/cells/SP8Cell/nodes/SP8Node/servers
   280              /BBOASR4A/was.env.
15.51.44 STC07693  +BBOM0001I wlm_dynapplenv_single_server: 1.
15.51.44 STC07693  +BBOM0001I      wlm_maximumSRCount: 1.
15.51.44 STC07693  +BBOM0001I      wlm_minimumSRCount: 1.
15.51.44 STC07693  +BBOM0001I                ECA_port: NOT SET, DEFAULT=0.
15.51.44 STC07693  +BBOO0024I ERRORS WILL BE WRITTEN TO CERR FOR JOB BBOASR4S.
15.51.44 STC07693  ITT122I CTRACE WRITER BBOWTR IS NOT STARTED.
                            ISSUE COMMAND TRACE CT,WTRSTART=BBOWTR.
15.51.50 STC07693  +BBOJ0011I JVM Build is J2RE 1.4.1 IBM z/OS Persistent Reusable VM  287
   287              build cm1411-20040318 (JIT enabled: jitc).
15.51.50 STC07693  +BBOJ0051I PROCESS INFORMATION: STC07693/BBOASR4S, ASID=118(0x76),  288
   288              PID=16777320(0x1000068)
15.51.50 STC07693  +BBOJ0077I                    java.vendor = IBM Corporation
15.51.50 STC07693  +BBOJ0077I                        os.name = z/OS
15.51.50 STC07693  +BBOJ0077I            sun.boot.class.path =  291
   291              /usr/lpp/java/J1.4/lib/core.jar:/usr/lpp/java/J
15.51.50 STC07693  +BBOJ0077I  292
   292              1.4/lib/graphics.jar:/usr/lpp/java/J1.4/lib/sec
15.51.50 STC07693  +BBOJ0077I  293
   293              urity.jar:/usr/lpp/java/J1.4/lib/ibmjssefips.ja
15.51.50 STC07693  +BBOJ0077I  294
   294              r:/usr/lpp/java/J1.4/lib/server.jar:/usr/lpp/ja
15.51.50 STC07693  +BBOJ0077I  295
   295              va/J1.4/lib/xml.jar:/usr/lpp/java/J1.4/lib/char
15.51.50 STC07693  +BBOJ0077I  296
   296              sets.jar:/usr/lpp/java/J1.4/classes
15.51.50 STC07693  +BBOJ0077I   java.vm.specification.vendor = Sun Microsystems Inc.
15.51.50 STC07693  +BBOJ0077I           java.runtime.version = 1.4.1
15.51.50 STC07693  +BBOJ0077I security.use.localos.authorization = true
15.51.50 STC07693  +BBOJ0077I                      user.name = OMVS
15.51.50 STC07693  +BBOJ0077I                  java.compiler = jitc
15.51.50 STC07693  +BBOJ0077I               was.install.root = /WebSphere/V5R1M0/AppServer
15.51.50 STC07693  +BBOJ0077I    com.ibm.websphere.SpecLevel = 1.3
15.51.50 STC07693  +BBOJ0077I                  user.language = en
15.51.50 STC07693  +BBOJ0077I          sun.boot.library.path = /usr/lpp/java/J1.4/bin
15.51.50 STC07693  +BBOJ0077I com.ibm.ws390.INTERNAL.plexname = MONOSP8
15.51.50 STC07693  +BBOJ0077I               ws390.servername = BBOASR4A
15.51.50 STC07693  +BBOJ0077I                   java.version = 1.4.1
15.51.50 STC07693  +BBOJ0077I                  user.timezone = GMT
15.51.50 STC07693  +BBOJ0077I                    server.root = /WebSphere/V5R1M0/AppServer
15.51.50 STC07693  +BBOJ0077I            sun.arch.data.model = 32
15.51.50 STC07693  +BBOJ0077I             java.endorsed.dirs =  312
   312              /usr/lpp/java/J1.4/lib/endorsed
```

```
15.51.50 STC07693  +BBOJ0077I             file.encoding.pkg = sun.io
15.51.50 STC07693  +BBOJ0077I                file.separator = /
15.51.50 STC07693  +BBOJ0077I       java.specification.name = Java Platform API  315
   315              Specification
15.51.50 STC07693  +BBOJ0077I             traceSettingsFile =  316
   316              config/cells/SP8Cell/nodes/SP8Node/servers/BBOA
15.51.50 STC07693  +BBOJ0077I                                 SR4A/trace.dat
15.51.50 STC07693  +BBOJ0077I            java.class.version = 48.0
15.51.50 STC07693  +BBOJ0077I                     java.home = /usr/lpp/java/J1.4
15.51.50 STC07693  +BBOJ0077I                  java.vm.info = J2RE 1.4.1 IBM z/OS  320
   320              Persistent Reusable VM buil
15.51.50 STC07693  +BBOJ0077I                                 d cm1411-20040318 (JIT  321
   321              enabled: jitc)
15.51.50 STC07693  +BBOJ0077I com.ibm.CORBA.iiop.noLocalCopies = false
15.51.50 STC07693  +BBOJ0077I                    os.version = 01.03.00
15.51.50 STC07693  +BBOJ0077I               was.history.dir = /tmp
15.51.50 STC07693  +BBOJ0077I                path.separator = :
15.51.50 STC07693  +BBOJ0077I               java.vm.version = 1.4.1
15.51.50 STC07693  +BBOJ0077I java.util.prefs.PreferencesFactory =  327
   327              java.util.prefs.FileSystemPreferencesFactory
15.51.50 STC07693  +BBOJ0077I     java.protocol.handler.pkgs = com.ibm.crypto.provider
15.51.50 STC07693  +BBOJ0077I           java.awt.printerjob = sun.print.PSPrinterJob
15.51.50 STC07693  +BBOJ0077I          java.security.policy =  330
   330              /WebSphere/V5R1M0/AppServer/properties/server.p
15.51.50 STC07693  +BBOJ0077I                                 olicy
15.51.50 STC07693  +BBOJ0077I       sun.io.unicode.encoding = UnicodeBig
15.51.50 STC07693  +BBOJ0077I           was.repository.root =  333
   333              /WebSphere/V5R1M0/AppServer/config
15.51.50 STC07693  +BBOJ0077I                java.assistive = ON
15.51.50 STC07693  +BBOJ0077I com.ibm.ws390.INTERNAL.DriverPath = null
15.51.50 STC07693  +BBOJ0077I                     user.home = /
15.51.50 STC07693  +BBOJ0077I     java.specification.vendor = Sun Microsystems Inc.
15.51.50 STC07693  +BBOJ0077I             java.library.path =  338
   338              /usr/lpp/java/J1.4/bin:/usr/lpp/java/J1.4/bin/c
15.51.50 STC07693  +BBOJ0077I  339
   339              lassic:/WebSphere/V5R1M0/AppServer/lib
15.51.50 STC07693  +BBOJ0077I               java.vendor.url = http://www.ibm.com/
15.51.50 STC07693  +BBOJ0077I                    SERVERNAME = CLUSTER1
15.51.50 STC07693  +BBOJ0077I                java.vm.vendor = IBM Corporation
15.51.50 STC07693  +BBOJ0077I             platform.notASCII = true
15.51.50 STC07693  +BBOJ0077I              java.fullversion = J2RE 1.4.1 IBM z/OS  344
   344              Persistent Reusable VM buil
15.51.50 STC07693  +BBOJ0077I                                 d cm1411-20040318 (JIT  345
   345              enabled: jitc)
15.51.50 STC07693  +BBOJ0077I             java.runtime.name = Java(TM) 2 Runtime  346
   346              Environment, Standard Editio
15.51.50 STC07693  +BBOJ0077I                                 n
15.51.50 STC07693  +BBOJ0077I               java.class.path =  348
   348              /WebSphere/V5R1M0/AppServer/properties:/WebSphe
15.51.50 STC07693  +BBOJ0077I  349
   349              re/V5R1M0/AppServer/lib/bootstrap.jar:/WebSpher
15.51.50 STC07693  +BBOJ0077I  350
   350              e/V5R1M0/AppServer/lib/urlprotocols.jar:/WebSph
15.51.50 STC07693  +BBOJ0077I  351
   351              ere/V5R1M0/AppServer/lib/j2ee.jar:/WebSphere/V5
15.51.50 STC07693  +BBOJ0077I  352
   352              R1M0/AppServer/lib/lmproxy.jar
15.51.50 STC07693  +BBOJ0077I com.ibm.ws390.INTERNAL.ServerName = CLUSTER1
15.51.50 STC07693  +BBOJ0077I                   ws.ext.dirs =  354
   354              /usr/lpp/java/J1.4/lib:/WebSphere/V5R1M0/AppSer
15.51.50 STC07693  +BBOJ0077I  355
   355              ver/classes:/WebSphere/V5R1M0/AppServer/lib:/We
15.51.50 STC07693  +BBOJ0077I  356
   356              bSphere/V5R1M0/AppServer/lib/ext:/WebSphere/V5R
15.51.50 STC07693  +BBOJ0077I  357
   357              1M0/AppServer:/WebSphere/V5R1M0/AppServer/web/h
15.51.50 STC07693  +BBOJ0077I  358
```

```
   358             elp:/WebSphere/V5R1M0/AppServer/deploytool/itp/
 15.51.50 STC07693  +BBOJ0077I  359
   359             plugins/com.ibm.etools.ejbdeploy/runtime
 15.51.50 STC07693  +BBOJ0077I     java.vm.specification.name = Java Virtual Machine  360
   360             Specification
 15.51.50 STC07693  +BBOJ0077I  java.vm.specification.version = 1.0
 15.51.50 STC07693  +BBOJ0077I            java.awt.headless = true
 15.51.50 STC07693  +BBOJ0077I            user.install.root = /WebSphere/V5R1M0/AppServer
 15.51.50 STC07693  +BBOJ0077I               java.io.tmpdir = /tmp
 15.51.50 STC07693  +BBOJ0077I          ibm.system.encoding = Cp1047
 15.51.50 STC07693  +BBOJ0077I                      os.arch = 390
 15.51.50 STC07693  +BBOJ0077I         java.awt.graphicsenv =  367
   367             sun.awt.X11GraphicsEnvironment
 15.51.50 STC07693  +BBOJ0077I               ws390.nodename = SP8Node
 15.51.50 STC07693  +BBOJ0077I                java.ext.dirs =  369
   369             /usr/lpp/java/J1.4/lib/ext:/WebSphere/V5R1M0/Ap
 15.51.50 STC07693  +BBOJ0077I                                pServer/java/jre/lib/ext
 15.51.50 STC07693  +BBOJ0077I                     user.dir = /WebSphere/V5R1M0/AppServer
 15.51.50 STC07693  +BBOJ0077I com.ibm.ws390.INTERNAL.ServerInstanceName = BBOASR4
 15.51.50 STC07693  +BBOJ0077I         com.ibm.itp.location =  373
   373             /WebSphere/V5R1M0/AppServer/bin
 15.51.50 STC07693  +BBOJ0077I               line.separator =   x'0A'
 15.51.50 STC07693  +BBOJ0077I                 java.vm.name = Classic VM
 15.51.50 STC07693  +BBOJ0077I java.security.auth.login.config =  376
   376             /WebSphere/V5R1M0/AppServer/properties/wsjaas.c
 15.51.50 STC07693  +BBOJ0077I                                onf
 15.51.50 STC07693  +BBOJ0077I   com.ibm.websphere.ServerType = J2EE
 15.51.50 STC07693  +BBOJ0077I                file.encoding = ISO8859-1
 15.51.50 STC07693  +BBOJ0077I            smpe.install.root = /usr/lpp/zWebSphere/V5R1M0
 15.51.50 STC07693  +BBOJ0077I     java.specification.version = 1.4
 15.52.07 STC07693  +BBOO0222I ADMN0015I: AdminService initialized
 15.52.10 STC07693  +BBOO0222I SECJ0215I: Successfully set JAAS login provider  383
   383             configuration class to com.ibm.ws.security.auth.login.Configuration.
 15.52.41 STC07693  +BBOO0222I ACWA0010I: WorkArea service ready on BBOASR4A
 15.52.56 STC07693  +BBOO0222I WSVR0200I: Starting application: adminconsole
 15.52.58 STC07693  +BBOO0222I SRVE0161I: IBM WebSphere Application Server - Web Container.  386
   386             Copyright IBM Corp. 1998-2002
 15.52.58 STC07693  +BBOO0222I SRVE0162I: Servlet Specification Level: 2.3
 15.52.58 STC07693  +BBOO0222I SRVE0163I: Supported JSP Specification Level: 1.2
 15.52.58 STC07693  +BBOO0222I SRVE0169I: Loading Web Module: adminconsole.
 15.53.10 STC07693  +BBOO0222I WSVR0221I: Application started: adminconsole
 15.53.10 STC07693  +BBOO0222I WSVR0200I: Starting application: ivtApp
 15.53.11 STC07693  +BBOO0222I SRVE0169I: Loading Web Module: IVT Application.
 15.53.12 STC07693  +BBOO0222I WSVR0221I: Application started: ivtApp
 15.53.12 STC07693  +BBOO0222I WSVR0200I: Starting application: JMSPrim_EMMDB.ear
 15.53.34 STC07693  +BBOO0222I WSVR0221I: Application started: JMSPrim_EMMDB.ear
 15.53.35 STC07693  +BBOO0222I SRVE0171I: Transport http is listening on port 80.
 15.53.35 STC07693  +BBOO0222I SRVE0171I: Transport https is listening on port 443.
 15.53.36 STC07693  +BBOO0222I WSVR0001I: Server SERVANT PROCESS BBOASR4A open for  428
   428             e-business
 15.53.36 STC07693  +BBOO0020I INITIALIZATION COMPLETE FOR WEBSPHERE FOR Z/OS SERVANT  429
   429             PROCESS BBOASR4.
 15.53.36 STC07693  +BBOO0248I INITIALIZATION COMPLETE FOR WEBSPHERE FOR Z/OS SERVANT  430
```

Internal tracing tips for WebSphere for z/OS

WebSphere traces can be extremely helpful in detecting and diagnosing problems. By properly setting trace options, you can capture the information needed to detect problems without significant performance overhead.

- Ensure that you are not collecting more diagnostic data than you need.

  You should check your WebSphere for z/OS tracing options to ensure that ras_trace_defaultTracingLevel=0 or 1, and that ras_trace_basic and ras_trace_detail are not set.

  How to view or set: Use the WebSphere administrative console:

  1. Click **Environment > Manage WebSphere Variables**.

2. On the Configuration Tab check for any of these variables in the name field and observe the variable setting in the value field.
3. To change or set a variable, specify the variable in the name field and specify the setting in the value field. You can also describe the setting in the description field on this tab.

- If you use any level of tracing, including ras_trace_defaultTracingLevel=1, ensure that you set ras_trace_outputLocation to BUFFER.

  ras_trace_defaultTracingLevel=1 will write exceptions to the trace log as well as to the ERROR log.
  - It is best to trace to CTRACE.

    If you are tracing to sysprint with ras_trace_defaultTracingLevel=3, you may experience an almost 100% throughput degradation. If you are tracing to CTRACE, however, you may only experience a 15% degradation in throughput.
- Set the ras_trace_BufferCount=4 and ras_trace_BufferSize=128.

  This will get 512KB of storage for the trace buffers (the minimum allowed) and reduce memory requirements.
- Make sure you disable JRAS tracing.

  To do this, look for the following lines in the trace.dat file pointed to by the JVM properties file:

  ```
  com.ibm.ejs.*=all=disable

  com.ibm.ws390.orb=all=disable
  ```

  Ensure that both lines are set to **=disable** or delete the two lines altogether.

  Note: If ras_trace_outputLocation is set, you may be tracing and not know it.

Location of executable programs tips for z/OS

The next thing to review in the configuration is where your program code is located. IBM recommends that you install as much of the WebSphere for z/OS code in LPA as is reasonable, and the remainder in the linklist. This ensures that you have eliminated any unnecessary steplibs which can adversely affect performance. If you must use STEPLIBs, verify that any `STEPLIB DD`s in the controller and servant procs do not point to any unnecessary libraries. Refer to “UNIX System Services (USS) tuning tips for z/OS” on page 99 for USS shared file system tuning considerations.

If you choose to not put most of the runtime in LPA, you may find that your processor storage gets a bigger workout as the load increases. At a minimum, WebSphere for z/OS will start three address spaces, so that any code that is not shared will load three copies rather than one. As the load increases, many more servants may start and will contribute additional load on processor storage.

Review the `PATH` statement to ensure that only required programs are in the PATH and that the order of the PATH places frequently-referenced programs in the front.

Security tuning tips for z/OS

As a general rule, two things happen when you increase security: the cost per transaction increases and throughput decreases.

By default, WebSphere for z/OS runs with security off. With security turned off there is virtually no overhead. However, we recommend that you run with security enabled, even though the runtime will always incur a small price to collect and carry the security credential information for users and the server. Compared to no security, global security for a sample application took about 13%. Compared to no security, java2 security took about 18%. Enabling a single role on every method in a sample application with global and java2 security costs about 38%.

- When a SAF (RACF or equivalent) class is active, the number of profiles in a class will affect the overall performance of the check.

  Placing these profiles in a (RACLISTed) memory table will improve the performance of the access checks. Audit controls on access checks also affect performance. Usually, you audit failures and not successes. Audit events are logged to DASD and will increase the overhead of the access check. Since all the security authorization checks are done with SAF (RACF or equivalent), you can choose to enable and disable SAF classes to control security. A disabled class will cost a negligible amount of overhead.
- Use a minimum number of EJBROLEs on methods.

  If you are using EJBROLEs, specifying more roles on a method will lead to more access checks that need to be executed and a slower overall method dispatch. If you are not using EJBROLEs, do not activate the class.
- If you do not need Java 2 security, you should disable it.

  For instructions on how to disable Java 2 security, refer to Configuring Java 2 security
- Use the lowest level of authorization consistent with your security needs.

  You have several options when dealing with authentication:
  - **Local authentication:** Local authentication is the fastest type because it is highly optimized.
  - **UserID and password authentication:** Authentication that utilizes a userID and password has a high first-call cost and a lower cost with each subsequent call.
  - **Kerberos security authentication:** We have not adequately characterized the cost of kerberos security yet.
  - **SSL security authentication:** SSL security is notorious in the industry for its performance overhead. Luckily, there is a lot of assists available from hardware to make this reasonable on z/OS.
- If using SSL, select the lowest level of encryption consistent with your security requirements.

  WebSphere allows you to select which cipher suites you use. The cipher suites dictate the encryption strength of the connection. The higher the encryption strength, the greater the impact on performance. For more information on SSL, refer to Secure Sockets Layer.

Servlet and EJB integrated runtime tuning tips for z/OS

If you are running with an HTTP server (DGW) as well as WebSphere on your z/OS system, you can run servlets and JSPs under the HTTP server or you can run servlets and JSPs under WebSphere in an environment called the servlet/EJB integrated runtime. Normally, servlets and JSPs are run under the HTTP server since this is more efficient. However, the integrated runtime provides a more powerful environment and the performance impact of using the integrated runtime is only significant if servlets/JSPs perform only trivial functions.

If you are running just a servlet, the integrated runtime may not initially show an improvement in performance. However, when a servlet is calling on an EJB, it will benefit greatly from the integrated runtime. Essentially, the integrated runtime will convert the remote method calls to local in-process EJB invocations (which are much faster).

The Servlet/EJB integrated runtime provides functions such as access to transactional resource managers and it is the only way to get true J2EE compliance with WebSphere for z/OS.

WebSphere Application Server tuning tips for use with DB2

Prepared statement caching effects on DB2 for OS/390 JDBC cursor objects

WebSphere Application Server uses JDBC prepared statement caching as a performance enhancing feature. If you are using this feature together with DB2 for OS/390, be aware of the potential impact on the number of DB2 JDBC cursor objects available.

When you obtain a *ResultSet* object by executing a *PreparedStatement* object, a DB2 JDBC cursor object is bound to it until the corresponding DB2 prepared statement is closed. This happens when the DB2 *Connection* object is released from the WebSphere Application Server connection pool. From an application perspective, the result set, prepared statement, and connection are each closed in turn. However, the underlying DB2 *Connection* is pooled by the WebSphere Application Server, the underlying DB2 *PreparedStatement* is cached by the application server, and each underlying DB2 JDBC cursor object associated with each *ResultSet* created on this *PreparedStatement* object is not yet freed.

Each *PreparedStatement* object in the cache can have one or more result sets associated with it. If a result set is opened and not closed, even though you close the connection, that result set is still associated with the prepared statement in the cache. Each of the result sets has a unique JDBC cursor attached to it. This cursor is kept by the statement and is not released until the prepared statement is cleared from the WebSphere Application Server cache.

If there are more of the cached statements than there are cursors, eventually the execution of a *PreparedStatement* object results in the following exception:

```
java.sql.SQLException: DB2SQLJJDBCProfile Error: No more JDBC Cursors without hold
```

Some WebSphere Application Server tuning suggestions to help avoid this problem are:

1. Decrease the *statement cache size* setting on the DB2 for OS/390 data source definition. Setting this value to zero (0) eliminates statement caching, but causes a noticeable performance impact.
2. Decrease the *minConnections* connection pool setting on the DB2 for OS/390 data source definition.
3. Decrease the *Aged Timeout* connection pool setting on the DB2 for OS/390 data source definition. However, it is **NOT** recommended that you set this to zero (0), as this disables the Aged Timeout function.

Tuning for J2EE applications

Steps involved in tuning the J2EE applications performance include:

- “Topology planning and performance” on page 124
- “J2EE container and applications” on page 125
- “J2EE application programming tips” on page 125
- “Tuning for SOAP” on page 126
- “Configuring MDB throttle support” on page 128

Topology planning and performance

Topology can have a significant effect on WebSphere performance. This article describes some of the topology considerations you should be aware of when configuring and installing WebSphere for z/OS.

- Single server or multiple servers?

  WebSphere for z/OS gives you the ability to install your application either in a single server or spread it across multiple servers. There are many reasons for partitioning your application. However, for performance, placing your application all in the same server will always provide better performance than partitioning it. If you do choose to partition your application across servers, you will get better performance if there are at least replica servers on each system in the sysplex. The WebSphere for z/OS runtime will try to keep calls local to the system if it can, which will, for example, use local interprocess calls rather than sockets.
- One tran or multiple trans?

  You also have a choice of running server regions with an isolation policy of one tran per server region or multiple trans per server region. From a performance perspective, running more threads in a server region will consume less memory but at the cost of thread contention. This contention is application-dependent. We generally recommend the use of multiple trans unless you run into contention problems.

  Specify the threads setting using the *server_region_workload_profile*. The variables include:
  – ISOLATE - sets the value to 1 thread.
  – CPUBOUND
  – IOBOUND - default
  – LONGWAIT - 40

  The thread value increases with each variable to the maximum number available with the LONGWAIT setting (40). For more information refer to ORB services advanced settings

  Note: Please see the ″Servers″ section in the WebSphere Application Server for z/OS V5.0 information center, access to which can be obtained through the WebSphere forz/OS library Web site http://www.ibm.com/software/webservers/appserv/zos_os390/library.html for more information on ORB services advanced settings.
- Local client or remote client?

  On a local client, the client and the optimized communication are done on the same system. This has some additional client CPU costs but less communication cost. On a remote client, the client cost is replaced by the additional communication overhead of sockets. The CPU cost on either system is almost equivalent. Latency is better for a local client than for a remote client, meaning you will get better response time with a local client.
- One copy of a server or many clones?

  You can define more than one copy of a server on a system. These copies are called clones. We have found slight improvements in performance when running with a couple of clones as opposed to just one (very large configuration). While there is some benefit, IBM does not recommend, at this time, the creation of replicated control regions for the sole purpose of improving performance. We do, however, recommend them for eliminating a single point of failure and for handling rolling upgrades without introducing an outage.

J2EE container and applications

In WebSphere for z/OS, there are several types of EJBs and several transaction policies supported. Selection of each type has performance implications. While we won't be able to give an exhaustive treatise on this yet, we will give some basic rules.

Enterprise bean development performance ramifications: There are three basic bean types in WebSphere for z/OS: session, entity, and message driven.

- Session beans

  Within a session bean in WebSphere for z/OS, there are stateless and stateful session beans.

  Stateless session bean
  : The lowest overhead type of bean. They are cheap to create, do very little automatically and, if not cleaned up by the application, will go away when the server terminates.

  Stateful session bean
  : The default for a stateful session bean is to not harden its state to a backing store except in the case of a controlled server shut down. In this configuration, a stateful bean is slightly more overhead than a stateless bean. The configuration overhead of hardening the stateful bean state at the end of each transaction has yet to be quantified.

- Entity beans

  In WebSphere Application Server for z/OS V5, entity beans come in two flavors: bean managed persistence (BMP) and container managed persistence (CMP).

  Since managing persistence is the responsibility of the bean in BMP, it really depends on the way the load and store is implemented whether a BMP is faster than a CMP. CMP beans manage persistence. The CMP bean implementation is highly optimized and will often produce better performance than a typical BMP bean. Additional improvements are expected which will add even more flexibility to CMP beans.

- Message driven beans (MDBs). See Message-driven beans - an overview.
- Marking a bean method with the `readonly` extended deployment descriptor will cause the runtime to avoid writing the state of the bean back out. Specify this value on a method when you know that the method does not update any attributes of the bean.

Transaction policy tuning tips: There are seven transaction policies in WebSphere for z/OS:

- TRANSACTION_REQUIRES
- TRANSACTION_REQUIRES_NEW
- TRANSACTION_SUPPORTS
- TRANSACTION_NOT_SUPPORTED
- TRANSACTION_BEAN_MANAGED
- TRANSACTION_NEVER
- TRANSACTION_MANDATORY

These transaction policies control how EJBs are associated with global or local transactions. Generally, local transactions are the fastest.

J2EE application programming tips

These programming tips relate to the following topics:

- **JavaServer pages (JSPs)**
  - Disable session state of JSPs.

    <%@ page language="java" contentType="text/html" session="false" %>

- By default, JSPs will save session state
  <%@ page language="java" contentType="text/html" %>
- Replace setProperties() calls in your JSPs with direct calls to the appropriate setxxx methods.

- **Java Database Connectivity (JDBC)**
  - Make sure you are at the current JDBC level.
  - Use prepared statements to allow dynamic statement cache of DB2 on z/OS.
  - Don't include literals in the prepared statements, use a parameter marker "?" to allow dynamic statement cache of DB2 on z/OS.
  - Use the right getxxx method by each data type of DB2.
  - Turn auto commit off when just read-only operations are performed.
  - Use explicit connection context objects.
  - When coding an iterator, you have a choice of named or positioned. For performance, we recommend positioned iterators.
  - Close prepared statements before reusing the statement handle to prepare a different SQL statement within the same connection.
  - As a bean developer, you have the choice of JDBC or SQLJ. JDBC makes use of dynamic SQL whereas SQLJ generally is static and uses pre-prepared plans. SQLJ requires an extra step to create and bind the plan whereas JDBC does not. SQLJ, as a general rule, is faster than JDBC.
  - With JDBC and SQLJ, you are better off writing specific calls that retrieve just what you want rather than generic calls that retrieve the entire row. There is a high per-field cost.

Tuning for SOAP

The Simple Object Access Protocol (SOAP) is a lightweight protocol which provides a mechanism to use XML for exchanging structured and typed information between peers in a decentralized, distributed environment.

- Specify Pass by reference in Object Request Broker service settings to pass parameters by reference instead of by value. This only applies if you are exposing an EJB as a web service. Enabling the Pass by reference setting results in the following specifications: protocol_iiop_no_local_copies=1 (in was.env), and -Dcom.ibm.CORBA.iiop.noLocalCopies=true (in servant.jvm.options). Refer to Object Request Broker service settings in administrative console.
- Make certain that all traces are disabled unless you are actively debugging a problem.
- When defining transaction policies for your application, specify TX_NOT_SUPPORTED and select local transactions. Local transactions perform better than global transactions because WebSphere is not required to coordinate commit scope over multiple resource managers.
- Avoid passing empty attributes or empty elements in SOAP messages. Do not include extraneous and unneeded data in SOAP messages. If you can use document/literal style web services invocation to batch requests into a single SOAP message, this is preferable to sending multiple individual SOAP messages. SOAP applications will perform better with smaller SOAP messages containing fewer XML elements and especially fewer XML attributes. The contents of SOAP messages must be serialized and parsed. These are expensive operations and should be minimized. In other words, it is preferable to send 1, 10KB message than 10, 1KB messages. However, very large messages (for example, over 200KB) may impact system resources like memory.
- You may need to increase the default Java heap size. SOAP and XML (DOM) are storage-intensive and small heap sizes may result in excessive Java garbage

collection. We found a heapsize of 256M (the default) was optimal for most test cases in the laboratory. You can monitor garbage collection using the verbose:gc Java directive.

- Insure TCP/IP send/receive buffers are large enough to hold the bulk of the xml messages that will be sent.
- Consider using a Document Model rather than the RPC model. It provides complete control over the format of the XML but requires additional programming effort.
- When using RPC-style messages, try to send strings if possible.
- Consider writing your own serializers and deserializers, avoiding reflection.

SOAP V 2.3 in WebSphere V 5.0.1:

One- to three-sentence description that will appear as the first paragraph of the finished article.

SOAP 2.3 has a number of enhancements over SOAP 2.2. They are described in full in http://ws.apache.org/soap/releases.html#v2.2. Some of the key enhancements in WebSphere V 5.0 are as follows:

- Support for document-oriented messaging
- Web Services Definition Language (WSDL) Version 1.1
- Universal Description, Discovery, and Integration, UDDI4J Version 2.0, logging
- Web Services Invocation Framework (WSIF)
- Web Services Caching
- Some minor SOAP performance enhancements
- Elimination of the Nagle Algorithm

Web Services is a more expensive protocol than HTTP or socket communication. It is best used where its benefits can be exploited. For example, for integration of decentralized distributed environment, where the clients have little knowledge of the application implementation. We do not recommend that programs running in the same JVM use Web Services as a means of communication or invocation. For obvious reasons, calling a method using SOAP generally has longer response time than other forms of invocation. XML parsing serialization/deserialization and network latency are all inhibitors to Web Services performance. There is no support for locally optimized invocation such that network protocols can be avoided when client and server are collocated.

We have observed, one of the most expensive operations in the processing of a SOAP message, whether by SOAP, AXIS or the Tech preview, is the deserialization of the inbound SOAP message. This step converts the message from an inbound string in wire format to an XML document. Therefore, if either the client or the server receives a large SOAP message, that entity normally has the highest CPU cost. We have found the CPU time to be similar for z/OS acting as either a SOAP server or a SOAP client as long as the inbound and outbound message sizes are comparable.

There are many forms of an XML message that are equivalent, for example, messages with additional white space are equivalent to messages with fewer blanks or spaces. Therefore, it is advisable to create SOAP messages without formatted nesting (pretty printing), which adds additional spaces. The XML parser must examine all characters, including blanks. Therefore, XML documents with additional blank characters will take longer to parse.

Document-oriented messaging, unlike RPC messaging, is not required to be synchronous. The UDDI Registry is a special purpose data base wherein, establishments can register the WSDL Service Interface Definition and the Service Implementation Definition for Web Services. WSDL describes the interfaces to the web service. This essentially makes them available to other establishments or business partners. WebSphere V 5.0 for z/OS also supports a Private UDDI Registry for private access to Web Services.

Configuring MDB throttle support

Message driven bean (MDB) throttle support controls the flow of work requests (WRs) into the WLM queue. The WRs result from messages being placed into a JMS destination. The MDB throttle allows or blocks a WR from being queued based on the maximum sessions set for the Listener Port and the current amount of in-flight requests. For more information on MDB throttle support, refer to "MDB throttle support" on page 130.

Configuring MDB throttle support involves listener port settings, connection factory settings, and MQ property settings. The proper setting of these properties will result in a configuration that handles message processing satisfactorily.

1. Set the listener port maximum sessions property. The value of this property determines the high threshold value (high threshold = maximum sessions) and is used by the throttle to decide when to block or allow requests.
   a. To view this administrative console page, click **Servers-> Application Servers->** ***application_server*****-> Message Listener Service-> Listener Ports->** ***listener_port***
   b. Set the maximum sessions property to the value you want the MDB throttle to use as its high threshold value. A good initial value for this setting is 100. The throttle also uses this value to compute the low threshold value. The actual number of server sessions is limited by the number of worker threads in the servant region and by the connection factory session pool max connections property.
2. Set the WebSphere MQ Queue Connection Factory properties.
   a. To view this administrative console page, click **Resources-> WebSphere MQ JMS Provider-> WebSphere MQ Queue Connection Factories (under additional properties)**
   b. Select the Connection Factory specified for the Listener Port.
   c. Under the Additional Properties, activate the Connection Pool panel.
   d. Set the Max Connections property for the Connection Pool. This property may be set to the total amount of connections that will be created. We recommend setting it to at least the number of worker threads available to the servant region.
   e. Under Additional Properties of the Connection Factory, activate the Session Pool panel.
   f. Set the Max Connections property for the Session Pool. This property may be set to the total amount of sessions that will be created or used concurrently from a single connection. This allows sessions to be readily available and reduces the probability of having to wait for a session to become available in order to do message processing. If the value is less than the Listener Port Maximum sessions property and the number of worker threads, then the throttle might request more sessions than there are available. This would result in one or more requests having to wait for a session to become free, which could have an undesirable effect on concurrent processing of requests.

3. Set the MQ related properties. The properties discussed in this section pertain to properties that have an effect on the number of messages in the destination and the number of connections to the queue manager. These properties include the MQ Backout Count Threshold (BOTHRESH), IDBACK, and IDFORE. For more information on these MQ settings, refer to:

   WebSphere MQ Script Command (MQSC) Reference Book (SC34-6055-03)

   WebSphere MQ Problem Determination Guide (GC34-6054-01).

   a. Set the MQ Backout Count Threshold (BOTHRESH) to a value less than the Listener Port Maximum retries property. If the BOTHRESH value is greater than the Listener Port Maximum retries value, then the Listener Port will be stopped for excessive message roll backs as in the case of poison messages, and the corrupted message will still remain in the destination.

      Setting this property to a value less than the Listener Port Maximum retries value prevents the same message from remaining in the destination after the message has been rolled back the specified number of times. Reaching this count threshold causes the message to be placed into the backout queue, which helps reduce the number of messages in the destination. Only the required number of Work Requests are created for the healthy messages, therefore improving resources utilization and performance.

   b. Set CTHREAD to the maximum number of connections from batch, CICS, IMS, and TSO tasks to a single instance of WebSphere MQ. This can be thought of as the total of connections available to WebSphere MQ. Make sure that the total of the IDBACK and IDFORE properties discussed in the next two substeps do not exceed this value.

   c. Set IDBACK to the maximum number of connections to a single instance of WebSphere MQ from batch or TSO background tasks. The number of connections to the queue manager impact performance. Choose the minimal number of connections that allows satisfactory message processing.

   d. Set IDFORE to the maximum number of connections to a single instance of WebSphere MQ from TSO foreground tasks. The number of connections to the queue manager impact performance. Choose the minimal number of connections that allows satisfactory message processing.

To set the MDB throttle to allow 100 messages to be queued before blocking message retrieval on a server with 40 Worker Threads, two connections (one for serving the MDBs and one per MDB instance for sending a reply message from the MDB's onMessage method), and a Listener Port maximum retries set to 5, set the following properties:

1. Set the Listener Port Maximum sessions property to 100. This automatically sets the throttle High Threshold to 100.
2. Set the Connection Factory's Connection Pool to have enough connections by setting the Max Connections property to the minimum of both the Maximum sessions and the Max Connections, 40 in this case.

   The number of worker threads limits the amount of server sessions available. Therefore, assuming the advanced ORB setting of LONGWAIT (40 threads), the server session pool will be set to allow at most 40 server sessions to be created.
3. Set the Max Connections property for the Connection Pool Connection Factory Session Pool to have enough sessions by setting the Max Connections property to the amount of concurrent sessions to be used during message processing.

   In this example, there can be at most 40 MDB instances, and a total of 1 session is needed per MDB instance, then the property should be set to 40.
4. Set the MQ BOTHRESH property to less than the Listener Port Maximum retries property.

5. Set the CTHREAD value higher than 40.

   For the CTHREAD, IDBACK, and IDFORE the total number of connections and how they are used must be taken into consideration. In this example, at most 40 connections could be made to WebSphere MQ from the Connection Factory, so CTHREAD should be higher than this value.

For debugging tips, refer to Best practices for debugging MDB throttle support.

MDB throttle support:

The reasoning behind any throttling mechanism is to control the flow of the resource at hand from one stage to another. Satisfactory message retrieval should be obtained by adjusting the rate at which message reference (MRs) are handled and work requests (WRs) are created. Without any type of control over the amount of WRs that are queued into the WLM queue, an excessive amount of WRs accumulate and are less likely to get processed by a minimal number of resources in the Servant Region. Control is based on available resources.

The messaging support provided by WebSphere for z/OS allows message processing to be split across regions. The messaging component in the WAS Controller Region is in charge of processing the Message References generated by MQ after messages are placed in the destination. The messaging component in the WAS Servant Region has the responsibility of retrieving the messages from the destination and deliver them to the MDB. WLM is used to queue work requests related to the Message References received in the Controller Region. This architecture poses problems when a large number of messages are placed in the destination queue.

Messaging flow. A message is sent to a destination by a JMS or MQ client. The destination is browsed for messages. Once the message is detected, a Message Reference (MR) is created and sent to the Message Reference Handler (MRH). The MRH calls the required service to queue the Work Request (WR) into the WLM queue. The work is dispatched in the Servant Region. The MR is consumed by the Listener causing the actual message in the destination to be retrieved. The message is presented to the MDB. The message is acknowledged and removed from the destination after the MDB is done processing it.

A considerably large number of messages sent to the destination causes at least the same amount of WRs to be queued into the WLM queue. If the resources allocated in the Servant Region are not sufficient to process the large amount of work presented, messages might get browsed two or more times after a certain amount of time. This worsens the situation by scheduling more WRs for the same messages, causing frequent message re-delivery.

Messaging flow with the throttle support. A message is sent to a destination. An MR is created and sent to the MRH. The MRH notifies the throttle of the message reference. The Throttle allows or blocks a WR from being queued based on the Maximum sessions set for the Listener Port and the current amount of in-flight requests.

When a high threshold is met, the throttle blocks queue browsing. Message processing continues and current requests are dispatched to available Servant regions. When the low threshold is met, the throttle allows more WRs to be queued.

Tuning applications

Application assembly tools are used to build J2EE components and modules into J2EE applications. Generally, assembling consists of defining application components and their attributes including enterprise beans, servlets and resource references. Many of these application configuration settings and attributes play an important role in the run-time performance of the deployed application.

- EJB modules
  - Entity bean Bean Cache - Activate at and Bean Cache - Load at settings
  - Method extensions Isolation level and Access intent settings
  - Container transactions assembly settings
- Web modules
  - Web modules assembly settings
    - Distributable
    - Reload interval
    - Reload enabled
- Web components
  - Load on startup

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 Licensing
IBM Corporation
500 Columbus Avenue
Thornwood, New York 10594 USA

Trademarks and service marks