IBM WebSphere Application Server Network
Deployment, Version 5

IBM

Monitoring and Troublshooting

Compilation date: November 22, 2002

Contents

Trademarks and service marks

The following terms are trademarks of IBM Corporation in the United States, other countries, or both:

- Everyplace
- iSeries
- IBM
- Redbooks
- ViaVoice
- WebSphere
- zSeries

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

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

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

IBM WebSphere Application Server Network Deployment, Version 5: Monitoring and Troublshooting

Chapter 1. Welcome to Monitoring and Troubleshooting

To help identify application performance problems, the product collects performance data, provides interfaces that allow external applications to monitor the performance data, and provides tools to display performance data for analysis.

Similar capabilities are provided for problem determination, such as 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 ″). This infrastructure is compatible with and extends the JSR-077 specification.

PMI provides several types of interfaces to access performance data. A new JMX API is introduced in this version, but the servlet and Java client interfaces are still available for compatibility with Versions 3.5.5+ and 4.0+. These PMI interfaces are used to create tools to help monitor and tune performance.

Performance data can be monitored and analyzed with:
* The Tivoli Performance Viewer (formerly called Resource Analyzer)
* Other Tivoli monitoring tools
* Your own or third-party-developed 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)″).

Performance Monitoring Infrastructure (PMI) Request Metrics

IBM WebSphere Application Server also collects data by timing requests as they travel through the product components. PMI Request Metrics logs time spent in major components, such as Web container, Enterprise bean container, and database. These data points are recorded in logs and can be written to Application Response Time (ARM) agents used by Tivoli monitoring tools.

For more information about PMI Request Metrics, see (″Measuring data requests (Performance Monitoring Infrastructure Request Metrics)″).

Problems - data and tools

The purpose of this section is to aid you in understanding why your enterprise application, application server, or the product itself is not working and to help you resolve the problem. Unlike performance tuning which focuses on solving problems associated with slow processes and less-than-optimal performance, problem determination focuses on finding solutions to functional problems.

The kind of problem you are encountering, and how much you already know about it, determine what steps you should take to resolve it:

- For tips on investigating common problems organized according to tasks, see ("Troubleshooting by task: what are you trying to do?").
- If you already have an error message and want to quickly look up its explanation and recommended response, see ("Message reference").
- For help in knowing where to find product error and warning messages, interpreting messages, and product log files, see ("Working with message logs").
- Difficult problems may require the use of tracing, which exposes the low-level flow of control and interactions between product components. For help in understanding and using traces, see ("Working with trace").
- For help in adding log and trace capability to your own application, see ("Programming with the JRas framework").
- For help in using product utilities to help you diagnose the problem, see ("Working with troubleshooting tools").
- To find out how to look up documented problems, common mistakes, product prerequisites, and other problem-determination information on the product Web site, or to obtain technical support from IBM, see ("Obtaining help from IBM").

Chapter 2. Monitoring performance

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.

Steps for this task

1. Enable PMI services through the administrative console.

   In order to monitor performance data through the PMI interfaces, you must first enable the performance monitoring services through the administrative console and restart the server.

2. Collect the data.

   The monitoring levels that determine which data counters are enabled can be set dynamically, without restarting the server. This can be done in one of three ways:

   a. Enable data collection through the administrative console.

   b. Enable performance monitoring services through Tivoli Performance Viewer (formerly Resource Analyzer).

   c. Enable performance monitoring services using the command line.

3. Monitor and analyze performance data.

   You can monitor and analyze data with several tools:

   a. Monitor performance data with Tivoli Performance Viewer.

      This tool is included with WebSphere Application Server.

   b. Monitor performance data with other Tivoli monitoring tools.

   c. Monitor performance data with user-developed monitoring tools.

      Write your own applications to monitor performance data.

   d. Monitor performance with third-party monitoring tools.

What to do next

WebSphere Application Server also collects data through PMI Request Metrics. This feature times requests as they travel through WebSphere Application Server components. For more information about PMI Request Metrics see "Measuring data requests (Performance Monitoring Infrastructure Request Metrics)" on page 71.

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.

As shown in the figure, 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, Java client or JMX client. WebSphere Application Server contains Tivoli Performance Viewer, a Java client which displays and monitors performance data. See the topics Monitoring performance with Tivoli Performance Viewer (formerly Resource Analyzer), Tivoli performance monitoring and management solutions, Third-party performance monitoring and

management solutions, and Developing your own monitoring applications for more information on monitoring tools.

HTTP
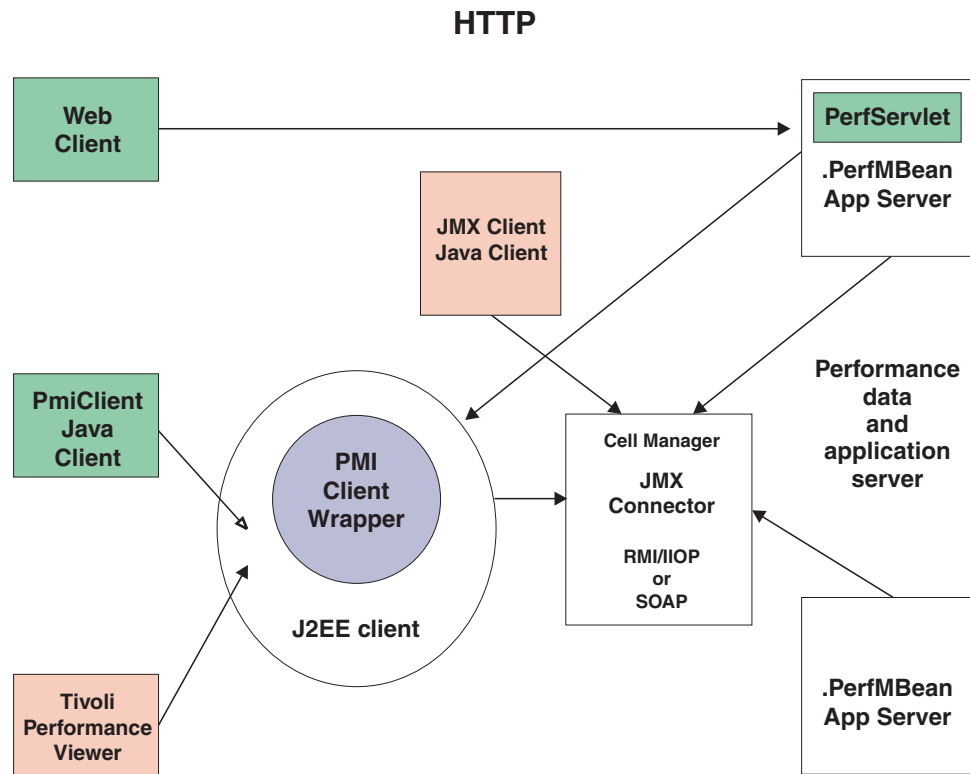
The figure shows the overall PMI architecture. On the right side, the server updates and keeps PMI data in memory. The left side displays a Web client, Java client and JMX client retrieving the performance data.

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.

Hiearchy of data collections used for performance reporting to Resource Analyzer
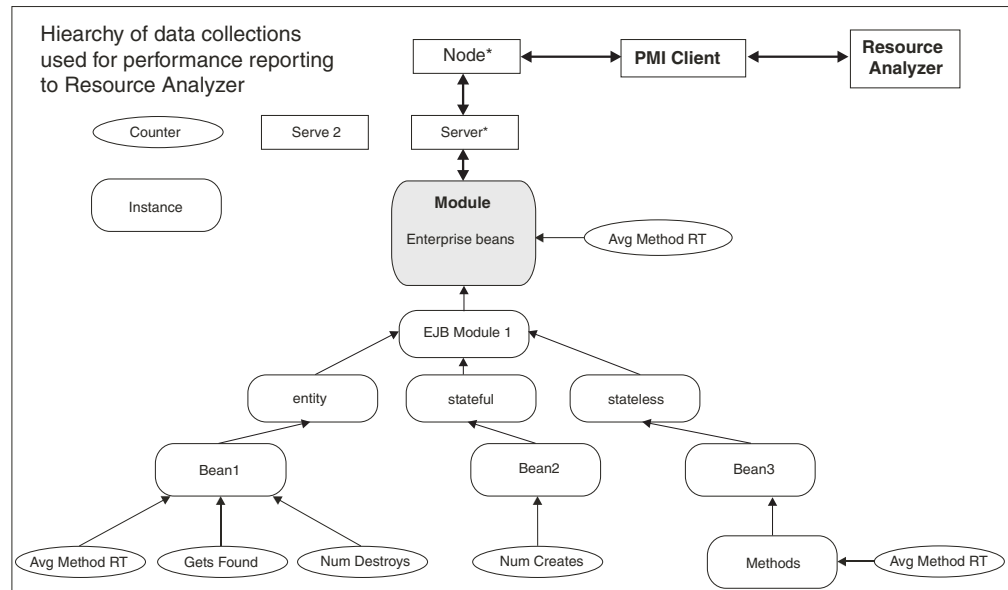
The Tivoli Performance Viewer, formerly the Resource Analyzer, organizes performance data in a centralized hierarchy of the following objects:

- **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, thread pools, transaction manager, Web applications, Object Request Broker (ORB), Workload Management (WLM), dynamic cache, and Web Services Gateway (WSGW).

- **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 will no longer include 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 will not be 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.

```
                        ModuleTreeRoot
                              |
     ┌────────────┬───────────┴──────────┬──────────────────┐
 EJBModule   ConnPoolModule          TranModule      WebAppModule      ....
     |            |                                        |
     |     ┌──────┴──────┐                          ┌──────┴──────┐
     |  dataSource 1  dataSource 2               webapp 1     webapp 2
     |                                              |
┌────┼────┐                                      servlets
bean 1 bean 2 bean 3                                 |
        |                                      ┌──────┴──────┐
     methods                                servlet 1   servlet 2
        |
 ┌──────┴──────┐
method 1   method 2
```
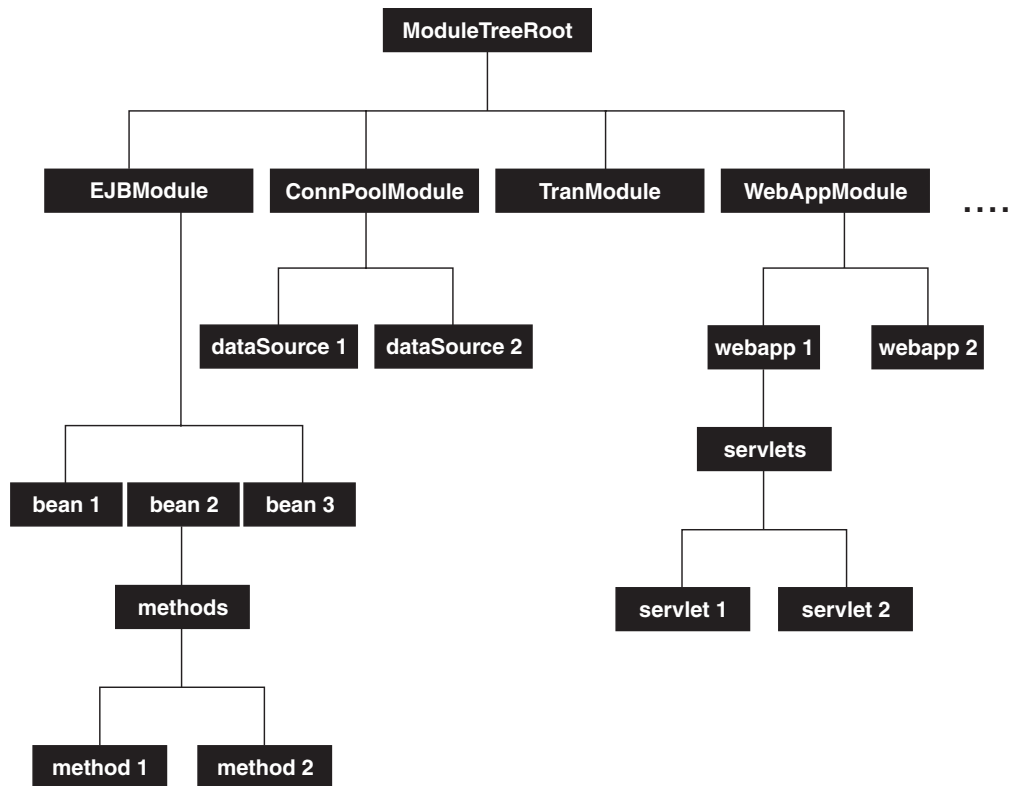
The following PMI modules are available to provide statistical data:

- Enterprise bean module, enterprise bean, methods in a bean

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

- JDBC connection pools

  Data counters for this category contain usage information about 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.

- J2C connection pool

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

- Java Virtual Machine API (JVM)

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

- Servlet session manager

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

- Thread pool

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

- Java Transaction API (JTA)

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

- Web applications, servlet

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

- Object Request Broker (ORB)

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

- Web Services Gateway (WSGW)

  Data counters for this category contain information for WSGW. Examples include the number of synchronous and asynchronous requests and responses.

- System data

  Data counters for this category contain information for a machine (node). Examples include the CPU utilization and memory usage. Note that this category is available at node level, which means it is only available at NodeAgent in the multiple servers version.

- Workload Management (WLM)

  Data counters for this category contain information for workload management. Examples include the number of requests, the number of updates and average response time.
- Dynamic cache

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

BeanModule data counters

Data counter definitions

Name	Description	Version	Granularity	Type	Level
creates	Number of times beans were created	3.5.5 and above	per home	CountStatistic	Low
removes	Number of times beans were removed	3.5.5 and above	per home	CountStatistic	Low
passivates	Number of times beans were passivated (entity and stateful)	3.5.5 and above	per home	CountStatistic	Low
activates	Number of times beans were activated (entity and stateful)	3.5.5 and above	per home	CountStatistic	Low
persistence loads	Number of times bean data was loaded from persistent storage (entity)	3.5.5 and above	per home	CountStatistic	Low
persistence stores	Number of times bean data was stored in persistent storage (entity)	3.5.5 and above	per home	CountStatistic	Low
instantiations	Number of times bean objects were instantiated	3.5.5 and above	per home	CountStatistic	Low
destroys	Number of times bean objects were freed	3.5.5 and above	per home	CountStatistic	Low
Num Ready Beans	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 above	per home	RangeStatistic	High
concurrent live	Number of concurrently live beans	3.5.5 and above	per home	RangeStatistic	High
avg method rsp time	Average response time in milliseconds on the bean methods (home, remote, local)	3.5.5 and above	per home	TimeStatistic	High
avg method rsp time for create	Average time in milliseconds a bean create call takes including the time for the load if any	5.0	per home	TimeStatistic	Medium

avg method rsp time for load	Average time in milliseconds for loading the bean data from persistent storage (entity)	5.0	per home	TimeStatistic	Medium
avg method rsp time for store	Average time in milliseconds for storing the bean data to persistent storage (entity)	5.0	per home	TimeStatistic	Medium
avg method rsp time for remove	Average time in milliseconds a bean entries call takes including the time at the database, if any	5.0	per home	TimeStatistic	Medium
total method calls	total number of method calls	3.5.5 and above	per home	CountStatistic	High
avg method rsp time for activation	Average time in milliseconds a beanActivate call takes including the time at the database, if any	5.0	per home	TimeStatistic	Medium
avg method rsp time for passivation	Average time in milliseconds a beanPassivate call takes including the time at the database, if any	5.0	per home	TimeStatistic	Medium
active methods	Number of concurrently active methods - num methods called at the same time.	3.5.5 and above	per home	TimeStatistic	High
Per method invocations	Number of calls to the bean methods (home, remote, local)	3.5.5 and above	per method/per home	CountStatistic	Max
Per method rsp time	Average response time in milliseconds on the bean methods (home, remote, local)	3.5.5 and above	per home	TimeStatistic	Max
Per method concurrent invocations	Number of concurrent invocations to call a method	5.0	per method/per home	RangeStatistic	Max
getsFromPool	Number of calls retrieving an object from the pool(entity and stateless)	3.5.5 and above	per home/object pool	CountStatistic	Low
getsFound	Number of times a retrieve found an object available in the pool (entity and stateless)	3.5.5 and above	per home/object pool	CountStatistic	Low
returnsToPool	Number of calls returning an object to the pool (entity and stateless)	3.5.5 and above	per home/object pool	CountStatistic	Low
returnsDiscarded	Number of times the returning object was discarded because the pool was full (entity and stateless)	3.5.5 and above	per home/object pool	CountStatistic	Low
drainsFromPool	Number of times the daemon found the pool was idle and attempted to clean it (entity and stateless)	3.5.5 and above	per home/object pool	CountStatistic	Low
avgDrainSize	Average number of objects discarded in each drain (entity and stateless)	3.5.5 and above	per home/object pool	TimeStatistic	Medium

avgPoolSize	Number of objects in the pool (entity and stateless)	3.5.5 and above	per home/object pool	RangeStatistic	High
messageCount	Number of messages delivered to the bean onMessage method (message driven beans)	5.0	per type	CountStatistic	Low
messageBackoutCount	Number of messages failed to be delivered to the bean onMessage method (message driven beans)	5.0	per type	CountStatistic	Low
serverSessionWait	Average time to obtain a ServerSession from the pool (message drive bean)	5.0	per type	TimeStatistic	Medium
serverSessionUsage	Percentage of server session pool in use (message driven)	5.0	per type	RangeStatistic	High

JDBC connection pool data counters

Data counter definitions

Name	Description	Version	Granularity	Type	Level
creates	Total number of connections created	3.5.5 and above	per connection pool	CountStatistic	Low
avg pool size	Average pool size	3.5.5 and above	per connection pool	BoundedRangeStatistic	High
free pool size	Average free pool size	5.0	per connection pool	BoundedRangeStatistic	High
allocates	Total number of connections allocated	3.5.5 and above	per connection pool	CountStatistic	Low
returns	Total number of connections returned	4.0 and above	per connection pool	CountStatistic	Low
avg waiting threads	Number of threads that are currently waiting for a connection	3.5.5 and above	per connection pool	RangeStatistic	High
connection pool faults	Total number of faults, such as, timeouts, in connection pool	3.5.5 and above	per connection pool	CountStatistic	Low
destroys	Number of times bean objects were freed	3.5.5 and above	per connection pool	CountStatistic	Low
avg wait time	Average waiting time in milliseconds until a connection is granted	5.0	per connection pool	TimeStatistic	Medium
avg time in use	Average time a connection is used	5.0	per connection pool	TimeStatistic	Medium
percent used	Average percent of the pool that is in use	3.5.5 and above	per connection pool	RangeStatistic	High

percent maxed	Average percent of the time that all connections are in use	3.5.5 and above	per connection pool	RangeStatistic	High
PreparedStmt cache discard count	Total number of statements discarded by the LRU algorithm of the prepared statement cache	4.0 and above	per connection pool	CountStatistic	Low
Number managed connections	Number of ManagedConnection objects in use	5.0	per connection factory	CountStatistic	Low
Number connections	Current number of connection objects in use	5.0	per connection factory	CountStatistic	Low
jdbcOperationTimer	Amount of time in milliseconds spent executing in the JDBC driver	5.0	per data source	TimeStatistic	Medium

J2C connection pool data counters

Data counter definitions

Name	Description	Version	Granularity	Type	Level
Number managed connections	Number of ManagedConnection objects in use	5.0	per connection factory	CountStatistic	Low
Number connections	Current number of connection objects in use	5.0	per connection factory	CountStatistic	Low
Number managed connections created	Total number of connections created	5.0	per connection factory	CountStatistic	Low
Number managed connections destroyed	Total number of connections destroyed	5.0	per connection factory	CountStatistic	Low
Number managed connections allocated	Total number of connections allocated	5.0	per connection factory	CountStatistic	Low
Number managed connections freed	Total number of connections freed	5.0	per connection factory	CountStatistic	Low
faults	Number of faults, such as timeouts, in connection pool	5.0	per connection factory	CountStatistic	Low
free pool size	Number of free connections in the pool	5.0	per connection factory	BoundedRangeStatistic	High
pool size	Pool size	5.0	per connection factory	BoundedRangeStatistic	High
concurrent waiters	Average number of threads concurrently waiting for a connection	5.0	per connection factory	RangeStatistic	High
Percent used	Average percent of the pool that is in use	5.0	per connection factory	RangeStatistic	High
Percent maxed	Average percent of the time that all connections are in use	5.0	per connection factory	RangeStatistic	High
Average wait time	Average waiting time in milliseconds until a connection is granted	5.0	per connection factory	TimeStatistic	Medium

| Average use time | Average time in milliseconds that connections are in use | 5.0 | per connection factory | TimeStatistic | Medium |

Java Virtual Machine data counters

Data counter definitions

Name	Description	Version	Granularity	Type	Level
Free memory	Free memory in JVM run time	3.5.5 and above	per Java Virtual Machine (JVM)	CountStatistic	Low
Used memory	Used memory in JVM run time	3.5.5 and above	per JVM	CountStatistic	Low
Total memory	Total memory in JVM run time	3.5.5 and above	per JVM	BoundedRangeStatistic	High
Up time	The amount of time the JVM is running	5.0	per JVM	CountStatistic	Low
Number garbage collection calls	Number of garbage collection calls. This counter is not available unless `-XrunpmiJvmpiProfiler` is set when starting the JVM.	4.0 and above	per JVM	CountStatistic	Max
Average time between garbage collection	Average garbage collection in seconds between two garbage collection. This counter is not available unless `-XrunpmiJvmpiProfiler` is set when starting the JVM.	4.0 and above	per JVM	TimeStatistic	Max
Average garbage collection duration	Average duration of a garbage collection. This counter is not available unless `-XrunpmiJvmpiProfiler` is set when starting the JVM.	4.0 and above	per JVM	TimeStatistic	Max
num waits for a lock	Number of times that a thread waits for a lock.This counter is not available unless `-XrunpmiJvmpiProfiler` is set when starting the JVM.	4.0 and above	per JVM	CountStatistic	Max
avg time waiting for lock	Average time that a thread waits for a lock. This counter is not available unless `-XrunpmiJvmpiProfiler` is set when starting the JVM.	4.0 and above	per JVM	TimeStatistic	Max
Number of objects allocated	Number of objects allocated in heap. This counter is not available unless `-XrunpmiJvmpiProfiler` is set when starting the JVM.	4.0 and above	per JVM	CountStatistic	Max
Number of objects found	Number of objects in heap. This counter is not available unless `-XrunpmiJvmpiProfiler` is set when starting the JVM.	4.0 and above	per JVM	CountStatistic	Max

Number of objects freed	Number of objects freed in heap. This counter is not available unless `-XrunpmiJvmpiProfiler` is set when starting the JVM.	4.0 and above	per JVM	CountStatistic	Max

Object Request Broker data counters

Data counter definitions

Name	Description	Version	Granularity	Type	Level
referenceLookupTime	The time (in milliseconds) to look up an object reference before method dispatch can be carried out	5.0	Object Request Broker (ORB)	TimeStatistic	Medium
numRequest	The total number of requests sent to the ORB	5.0	ORB	CountStatistic	Low
concurrentRequests	The number of requests that are concurrently processed by the ORB	5.0	ORB	RangeStatistic	High
processingTime	The time (in milliseconds) it takes a registered portable interceptor to run	5.0	per interceptor	TimeStatistic	Medium

Session data counters

Data counter definitions

Name	Description	Version	Granularity	Type	Level
createdSessions	Number of sessions created	3.5.5 and above	per web application	CountStatistic	Low
invalidatedSessions	Number of sessions invalidated	3.5.5 and above	per web application	CountStatistic	Low
sessionLifeTime	The average session lifetime	3.5.5 and above	per web application	TimeStatistic	Medium
activeSessions	The number of concurrently active sessions. A session is active if WebSphere is currently processing a request which uses that session.	3.5.5 and above	per web application	RangeStatistic	High
liveSession	The number of sessions that are currently cached in memory	5.0 and above	per web application	RangeStatistic	High
NoRoomForNewSession	Applies only to session in memory with AllowOverflow=false. The number of times that a request for a new session can not be handled because it would exceed the maximum session count.	5.0	per Web application	CountStatistic	Low

cacheDiscards	Number of session objects that have been forced out of the cache. (An 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
externalReadTime	Time (milliseconds) taken in reading the session data from persistent store. For multirow sessions, the metrics are for the attribute; for single row sessions, the metrics are for the whole session. Applicable only for persistent sessions. When using a JMS persistent store, the user has the choice of whether to serialize the data being replicated. If they choose not to serialize the data, the counter will not be available.	5.0	per Web application	TimeStatistic	Medium
externalReadSize	Size of session data read from persistent store. Applicable only for (serialized) persistent sessions; similar to externalReadTime above.	5.0	per Web application	TimeStatistic	Medium
externalWriteTime	Time (milliseconds) taken to write the session data from the persistent store. Applicable only for (serialized) persistent sessions. Similar to externalReadTime above.	5.0	per Web application	TimeStatistic	Medium
externalWriteSize	Size of session data written to persistent store. Applicable only for (serialized) persistent sessions. Similar to externalReadTime above.	5.0	per Web application	TimeStatistic	Medium
affinityBreaks	The number of requests received for sessions that were last accessed from another Web application. This can indicate failover processing or a corrupt plug-in configuration.	5.0	per Web application	CountStatistic	Low
serializableSessObjSize	The size in bytes of (the serializable attributes of) in-memory sessions. Only count session objects that contain at least one serializable attribute object. Note that a session may 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

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
invalidatedViaTimeout	The number of requests for a session that no CountStatistic exists, presumeably because the session timed out.	5.0	per Web application	CountStatistic	Low
attemptToActivateNot ExistentSession	Number of requests for a session that no longer exists, presumeably 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 data counters

Data counter definitions

Name	Description	Version	Granularity	Type	Level
Number global transactions begun	Total number of global transactions begun on server	4.0 and above	per transaction manager/server	CountStatistic	Low
Number global transactions involved	Total number of global trans involved on server (for example, begun and imported)	4.0 and above	per transaction manager/server	CountStatistic	Low
Number local transactions begun	Total number of local transactions begun on server	4.0 and above	per transaction manager/server	CountStatistic	Low
Active global transactions	Number of concurrently active global transactions	3.5.5 and above	per transaction manager/server	CountStatistic	Low
Active local transactions	Number of concurrently active local transactions	4.0 and above	per transaction manager/server	CountStatistic	Low
Global transactions duration	Average duration of global transactions	3.5.5 and above	per transaction manager/server	TimeStatistic	Medium
Local transaction duration	Average duration of local transactions	4.0 and above	per transaction manager/server	TimeStatistic	Medium
Local transactions before_completion time	Average duration of before_completion for local transactions	4.0 and above	per transaction manager or server	TimeStatistic	Medium
Global transaction commit time	Average duration of commit for global transactions	4.0 and above	per transaction manager/server	TimeStatistic	Medium
Global transaction prepare time	Average duration of prepare for global transactions	4.0 and above	per transaction manager/server	TimeStatistic	Medium
Local transaction before_completion time	Average duration of before_completion for local transactions	4.0 and above	per transaction manager/server	TimeStatistic	Medium
Local transaction commit time	Average duration of commit for local transactions	4.0 and above	per transaction manager/server	TimeStatistic	Medium
Number global transactions committed	Total number of global transactions committed	3.5.5 and above	per transaction manager/server	CountStatistic	Low

Number of global transactions rolled back	Total number of global transactions rolled back	3.5.5 and above	per transaction manager/server	CountStatistic	Low
Number global transactions optimized	Number of global transactions converted to single phase for optimization	4.0 and above	per transaction manager/server	CountStatistic	Low
Number of local transactions committed	Number of local transactions committed	4.0 and above	per transaction manager/server	CountStatistic	Low
Number of local transactions rolled back	Number of local transactions rolled back	4.0 and above	per transaction manager/server	CountStatistic	Low
Number of global transactions timed out	Number of global transactions timed out	4.0 and above	per transaction manager/server	CountStatistic	Low
Number of local transactions timed out	Number of local transactions timed out	4.0 and above	per transaction manager/server	CountStatistic	Low

ThreadPool data counters

Data counter definitions

Name	Description	Version	Granularity	Type	Level
Thread creates	Total number of threads created	3.5.5 and above	per thread pool	CountStatistic	Low
Thread destroys	Total number of threads destroyed	3.5.5 and above	per thread pool	CountStatistic	Low
Active threads	The number of concurrently active threads	3.5.5 and above	per thread pool	RangeStatistic	High
Pool size	Average number of threads in pool	3.5.5 and above	per thread pool	BoundedRangeStatistic	High
Percent maxed	Average percent of the time that all threads are in use	3.5.5 and above	per thread pool	RangeStatistic	High

Web application data counters

Data counter definitions

Name	Description	Version	Granularity	Type	Level
numLoadedServlets	Number of servlets that were loaded	3.5.5 and above	per Web application	CountStatistic	Low
numReloads	Number of servlets that were reloaded	3.5.5 and above	per Web application	CountStatistic	Low
totalRequests	Total number of requests a servlet processed	3.5.5 and above	per servlet	CountStatistic	Low

concurrentRequests	Number of requests that are concurrently processed	3.5.5 and above	per servlet	RangeStatistic	High
responseTime	The response time, in milliseconds, of a servlet request	3.5.5 and above	per servlet	TimeStatistic	Medium
numErrors	Total number of errors in a servlet or Java Server Page (JSP)	3.5.5 and above	per servlet	CountStatistic	Low

Workload Management data counters

Data counter definitions

Name	Description	Version	Granularity	Type	Level
numIncomingRequests	Total number of incoming IIOP requests to an application server	5.0	per server	CountStatistic	Low
numIncomingStrongAffinityRequests	Number of incoming IIOP requests to an application server that are based on a strong affinity. A strong affinity request is defined as a request that must be serviced by this application server because of a dependency that resides on the server. This request could not successfully be serviced on another member in the server cluster. In Version 5.0 ND edition, transactional affinity is the only example of a strong affinity	5.0	per server	CountStatistic	Low
numIncomingNonAffinityRequests	Number of incoming IIOP requests to an application server based on no affinity. This request was sent to this server based on workload management selection policies that were decided in the Workload Management (WLM) run time of the client.	5.0	per server	CountStatistic	Low
numIncomingNonWLMObjectRequests	Number of incoming IIOP requests to an application server that came from a client that does not have the WLM run time present or where the object reference on the client was flagged not to participate in workload management.	5.0	per server	CountStatistic	Low

Name	Description	Version	Granularity	Type	Level
numServerClusterUpdates	Number of times initial or updated server cluster data is sent to a server member from the deployment manager. This metric determines how often cluster information is being propagated.	5.0	per server	CountStatistic	Low
numOfWLMClientServiced	Number of WLM enabled clients that have been serviced by this application server.	5.0	per server	CountStatistic	Low
numOfConcurrentRequests	Number of remote IIOP requests currently being processed by this server	5.0	per server	RangeStatistic	High
serverResponseTime	The response time (in milliseconds) of IIOP requests being serviced by an application server. The response time is calculated based on the time the request is received to the time when the reply is sent back out.	5.0	per server	TimeStatistic	Medium
numOfOutgoingRequests	The total number of outgoing IIOP requests being sent from a client to an application server	5.0	per WLM	CountStatistic	Low
numClientClusterUpdates	The number of times initial or updated server cluster data is sent to a WLM enabled client from server cluster member. Use this metric to determine how often cluster information is being propagated.	5.0	per WLM	CountStatistic	Low
clientResponseTime	The response time (in milliseconds) of IIOP requests being sent from a client. The response time is calculated based on the time the request is sent from the client to the time the reply is received from the server.	5.0	per WLM	TimeStatistic	Medium

System data counters

Data counter definitions

Name	Description	Version	Granularity	Type	Level

percentCpuUsage	The average system CPU utilization taken over the time interval since the last reading. Because the first call is required to perform initialization, an invalid value such as 0 will be returned. All subsequent calls will return the expected value. On SMP machines, the value returned will be the utilization averaged over all CPUs.	5.0	per node	CountStatistic	Low
freeMemory	The amount of real free memory available on the system. Real memory that is not allocated is only a lower bound on available real memory, since many operating systems take some of the otherwise unallocated memory and use it for additional I/O buffering. The exact amount of buffer memory which can be freed up is dependent on both the platform and the application(s) running on it.	5.0	per node	CountStatistic	Low
avgCpuUsage	The average percentCpuUsage that is busy after the server is started	5.0	per node	TimeStatistic	Medium

Dynamic cache data counters

Data counter definitions

Name	Description	Version	Granularity	Type	Level
maxInMemoryCacheSize	Maximum number of in-memory cache entries	5.0	per server	CountStatistic	Low
inMemoryCacheSize	Current number of in-memory cache entries	5.0	per server	CountStatistic	Low
totalTimeoutInvalidation	Aggregate of template timeouts and disk timeouts	5.0	per server	CountStatistic	Low
hitsInMemory	Requests for this cacheable object served from memory	5.0	per template	CountStatistic	Low
hitsOnDisk	Requests for this cacheable object served from disk	5.0	per template	CountStatistic	Low
explicitInvalidations	Total explicit invalidation issued for this template	5.0	per template	CountStatistic	Low
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

timeoutInvalidations	Cache entries evicted from memory and/or disk because their timeout has expired	5.0	per template	CountStatistic	Low
Entries	Current number of cache entries created from this template. Refers to the per-template equivalent of totalCacheSize.	5.0	per template	CountStatistic	Low
hitsRemove	Requests for this cacheable object served from other Java Virtual Machines (JVM) in the cluster	5.0	per template	CountStatistic	Low
Misses	Requests for this cacheable object that were not found in the cache	5.0	per template	CountStatistic	Low
RequestFromClient	Requests for this cacheable object generated by applications running on the application server	5.0	per template	CountStatistic	Low
requestsFromJVM	Requests for this cacheable object generated by cooperating caches in this cluster	5.0	per template	CountStatistic	Low
explicitInvalidationsFromMemory	Explicit invalidations resulting in an entry being removed from memory	5.0	per template	CountStatistic	Low
explicitInvalidationsFromDisk	Explicit invalidations resulting in an entry being removed from disk	5.0	per template	CountStatistic	Low
explicitInvalidationsNoOp	Explicit invalidations received for this template where no corresponding entry exists	5.0	per template	CountStatistic	Low
explicitInvalidationsLocal	Explicit invalidations generated locally, either programmatically or by a cache policy	5.0	per template	CountStatistic	Low
explicitInvalidationsRemote	Explicit invalidations received from a cooperating JVM in this cluster	5.0	per template	CountStatistic	Low
remoteCreations	Entries received from cooperating dynamic caches	5.0	per template	CountStatistic	Low

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

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.

```
                        ┌─────────────────────────┐
                        │        Statistic         │
                        ├─────────────────────────┤
                        │ name: String             │
                        │ unit: String             │
                        │ descriptions: String     │
                        │ startTime: long          │
                        └─────────────────────────┘
```

Statistic

name: String
unit: String
descriptions: String
startTime: long

CountStatistic

count: long

BoundaryStatistic

upperBound: long
lowerBound: long

RangeStatistic

highWaterMark: long
lowWaterMark: long

TimeStatistic

count: long
maxTime: long
minTime: long
totalTime: long
RequestRate: double

BoundedRangeStatistic

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 PMI services through the administrative console

In order to monitor performance data through the PMI interfaces, you must first enable the performance monitoring services through the administrative console.

Steps for this task
1. Open the administrative console.
2. Click **Servers** > **Application Servers** in the console navigation tree.
3. Click *server*.
4. Click 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.

What to do next

While you are in the **Configuration** tab, you can only update the settings located in the **Configuration** tab. If you want to dynamically change the PMI level settings when the application server is running, you have to click the **Runtime** tab in step 4, instead of the **Configuration** tab, and make the changes there.

Enabling performance data collection through the administrative console

To enable data collection in the administrative console, select the Performance Monitoring Infrastructure (PMI) modules and levels that you want to monitor.

Steps for this task

1. Open the administrative console.
2. Click **Servers** > **Application Servers** in the console navigation tree.
3. Click *server*.
4. Click the **Runtime** tab.
5. Click **Performance Monitoring Service**.
6. Select the PMI modules and levels to set the **initial specification level** field.
7. Click **Apply** or **OK**.
8. Click **Save**.

What to do next

These changes will take effect immediately, but will not be persistent. Use the **Configuration** tab for a persistent change. See Enabling PMI services through the administrative console for more information about making a persistent change.

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.

Specifications
Specifies the PMI module and monitoring level that you have set.

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 using the command line

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

Steps for this task

1. Enable PMI services through the administrative console.

   Make sure to restart the application server.

2. Run the wsadmin command.

   Using **wsadmin**, you can invoke operations on Perf Mbean to obtain the PMI data, set or obtain PMI monitoring levels and enable data counters.

   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

- ThreadPool
- TransactionService
- WebModule
- Servlet
- WLMAppServer
- WebServicesService
- WSGW

Usage scenario

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 (split for publication):

```
beanModule=H:cacheModule=H:connectionPoolModule=H:j2cModule=H:jvmRuntimeModule=H:
orbPerfModule=H:servletSessionsModule=H:systemModule=H:threadPoolModule=H:
transactionModule=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

- 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 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

- get MBean query string - for example, server MBean (split for publication)

```
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 all the beans within the BeanModule hierarchy.

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 (split for publication)

```
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 separated by a space " ". You can then use the name as the String parameter in getStatsString method, for example (split for publication):

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

Enabling Java Virtual Machine Profiler Interface data reporting

To enable Java Virtual Machine Profiler Interface (JVMPI) data reporting for each individual application server:

Steps for this task

1. Open the administrative console.
2. Click **Servers** > **Application Servers** in the console navigation tree.
3. Click the application server for which JVMPI needs to be enabled.
4. Click **Process Definition**
5. Click the **Java Virtual Machine**.
6. Type *-XrunpmiJvmpiProfiler* in the **genericJvmArguments** field.
7. Set the JVM level to **MAX**.
8. Click **Apply** or **OK**.
9. Click **Save**.
10. Start the application server, or restart the application server if it is currently running.
11. Refresh the Tivoli Performance Viewer if you are using it.

Java Virtual Machine Profiler Interface

The Tivoli Performance Viewer leverages a Java Virtual Machine Profiler Interface (JVMPI) to enable more comprehensive performance analysis. This profiling tool enables the collection of information, such as data about garbage collection, and the Java virtual machine (JVM) API that runs the application server.

JVMPI is a two-way function call interface between the JVM API and an in-process profiler agent. The JVM API notifies the profiler agent of various events, such as heap allocations and thread starts. The profiler agent can activate or deactivate specific event notifications, based on the needs of the profiler.

JVMPI supports partial profiling by enabling the user to choose which types of profiling information to collect and to select certain subsets of the time during which the JVM API is active. JVMPI moderately increases the performance impact.

This function is available on the Windows, AIX, and Solaris platforms.

Monitoring performance with Tivoli Performance Viewer (formerly Resource Analyzer)

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

Tivoli Performance Viewer is a Graphical User Interface (GUI) performance monitor for WebSphere Application Server.

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

Steps for this task

1. Start the Tivoli Performance Viewer.
2. Set monitoring levels.
3. View summary reports.
4. (Optional) Store data to a log file.
5. (Optional) Replay a performance data log file.
6. (Optional) View and modify performance chart data.
7. (Optional) Scale the performance data chart display.
8. (Optional) Refresh data.
9. (Optional) Clear values from tables and charts.
10. (Optional) Reset counters to zero.

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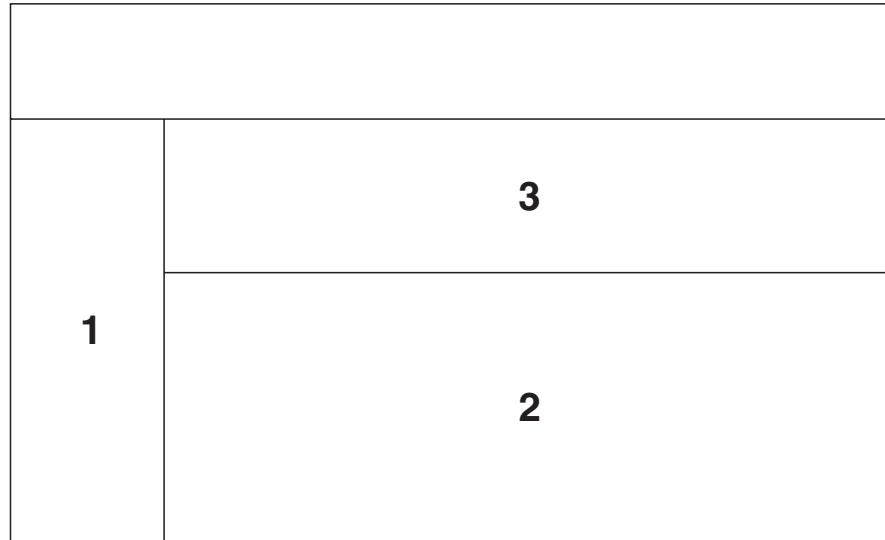
1 - Resource Selection Panel
2 - Counter Selection panel
3 - Viewing Counter (chart and table views)

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 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 supply 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

An alternative way to collect data is to use the Tivoli Performance Viewer, a Graphical User Interface (GUI) performance monitor shipped with WebSphere Application Server.

Steps for this task

1. Enable PMI services through the administrative console.
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</em>/bin` directory and run the `tperfviewer` script.

        You can specify the host and port in Windows NT and 2000 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.

        8879 is the default ND port for SOAP connector.

        9809 is the default ND port for RMI connector

    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 Remote Method Invocation (RMI) 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

Before you begin

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

To view monitoring settings:

Steps for this task

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 except enterprise bean method level data
   - **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**.

Results

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 data collection through the administrative console for more information.

Viewing summary reports

Before you begin

Summary reports are available for each application server. Before viewing reports, make sure data counters are enabled and monitoring levels are set properly.

The standard monitoring level will enable all reports except the report on enterprise bean methods. To enable enterprise bean method reports, use the custom monitoring setting and set the monitoring level to **Max** for the enterprise bean module.

To view the summary reports:

Steps for this task

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

<u>Before you begin</u>

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

To change the rate at which data is retrieved:

<u>Steps for this task</u>

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:

<u>Steps for this task</u>

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

<u>Before you begin</u>

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

To view data in a chart:

<u>Steps for this task</u>

1. Click a resource in the Resource Selection panel.
2. Click the **View Chart** tab in the Data Monitoring panel.

   Negative results display as zero (0). 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 counters so that the graphic displays enable meaningful comparisons between graphs of different counters. Follow these steps to manually adjust the scale:

<u>Steps for this task</u>

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.

<u>Results</u>

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. The value represents the default setting.
- 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 the variable is graphed as one and one-half times their actual values.
- 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 the variable is graphed as one-half its actual values.

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:

Steps for this task

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**.

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.

Clearing values from tables and charts

Before you begin

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:

Steps for this task

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

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:

Steps for this task
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. The XML format provides a flexibility that enables analysis by using third-party tools.
3. Click **OK**.

What to do next

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

Before you begin

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

To replay a log file, do the following:

Steps for this task

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 pathname in the **Log** field.
4. Click **Apply**.
5. Play the log by using the **Play** icon or click **Setting** > **Log Replay** > **Play**.

Results

By default, the data replays at the same rate it was collected or written to the log. If data is collected every minute, it displays every minute. 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** > **Fast Forward**.

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 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:

Steps for this task

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**.

Tivoli performance monitoring and management solutions

Tivoli offers the complete IBM solution for managing the extended WebSphere environment. For precise viewing of performance metrics, users can start with the Tivoli Performance Viewer, a complimentary tool shipped with WebSphere Application Server.

Tivoli also provides on-going production monitoring tools described below. For more information about Tivoli's solutions for WebSphere Application Server, see "Performance: Resources for learning" on page 78.

IBM Tivoli Monitoring for Web Infrastructure (ITMf WI). Provides best-practice monitoring of the key elements of WebSphere Application Server. This is the inside-out view, enabling administrators to quickly address problems before they impact end-users. Using Tivoli's advanced monitoring technology and predefined WebSphere best-practices, this tool quickly identifies problems, notifies appropriate personnel, and provides a solution. All monitoring data is displayed real-time with a health console displaying non-stop data. This same information can be uploaded to a common data warehouse for historical reporting.

IBM Tivoli Monitoring for Transaction Performance (ITMTP). Provides a unique monitoring perspective from that of the end-user. This is the outside-in view that verifies that end-to-end components provide a positive end-user experience. ITMTP monitors performance of actual and synthetic transactions, as well as verifying that the content delivered meets predefined guidelines.

Transaction performance includes total round trip response time, network latency, back-end response time and page render time. Additional granularity of transaction detail on the back-end is provided through Application Response Measurement instrumentation.

The ITM and ITMTP function by providing Web site performance monitoring, alerting customers to end user response time issues.

The ability to quickly find performance issues is key to maintaining a high performance Web site. This WebSphere Application Server release and the new ITMTP release combine to provide a new feature for analyzing performance problems. Using Synthetic Transaction Investigator (STI) from ITMTP, you can save key transactions and replay them later. ITMTP also collects the data provided by PMI Request Metrics through the Application Response Measurement (ARM) interface and correlates this information with the originating STI transaction. In the ITMTP real-time browser, the STI information links to the servlet and the enterprise bean response time data. The details regarding the overall transaction response time and response time for individual WebSphere Application Server components provide the ability to quickly identify performance problems.

Tivoli provides additional products for monitoring other key elements of the extended environment. For more information about Tivoli's solutions for WebSphere Application Server, see "Performance: Resources for learning" on page 78.

Developing your own monitoring applications

Before you begin

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. The PMI client interface is a Java interface that works with Version 3.5.5 and above. The servlet interface is perhaps the simplest, requiring minimal programming, as the output is XML.

Steps for this task
1. Developing your own monitoring application using Performance Monitoring Infrastructure client
2. Developing your own monitoring applications with PMI servlet
3. Running your new monitoring applications
4. Accessing Performance Monitoring Infrastructure data through the Java Management Extension interface.
5. Developing Performance Monitoring Infrastructure interfaces (Version 4.0)

Developing your own monitoring application using Performance Monitoring Infrastructure client

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

Steps for this task
1. Create an instance of PmiClient.

   This is used for all subsequent method calls.
2. Call the listNodes() and listServers(nodeName) methods to find all the nodes and servers in the WebSphere Application Server domain.

   The PMI client provides two sets of methods: one set in Version 5.0 and the other set inherited from Version 4.0. You can only use one set of methods. Do not mix them together.
3. Call listMBeans and listStatMembers to get all the available MBeans and MBeanStatDescriptors.
4. Call the getStats method to get the Stats object for the PMI data.
5. (Optional) The client can also call setStatLevel or getStatLevel to set and get the monitoring level. Use the MBeanLevelSpec objects to set monitoring levels.

What to do next

If you prefer to use the Version 4.0 version of the interface, the model is essentially the same, but the object types are different:
1. Create an instance of PmiClient.
2. Call the listNodes() and listServers(nodeName) methods to find all the nodes and servers in the WebSphere Application Server domain.
3. Call listMembers to get all the perfDescriptor objects.
4. Use the PMI client's get or gets method to get CpdCollection objects. These contain snapshots of performance data from the server. The same structure is maintained and its update method is used to refresh the data.
5. (Optional) The client can also call setInstrumentationLevel or getInstrumentationLevel to set and get the monitoring level.

Performance Monitoring Infrastructure client

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

You can use the PMI client package to write WebSphere Application Server clients that collect and display PMI data from servers.

Clients can be graphical user interfaces (GUIs) that display performance data in real-time, applications that monitor performance data and trigger different events according to the current values of the data, or any other application that needs to receive and process performance data.

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.

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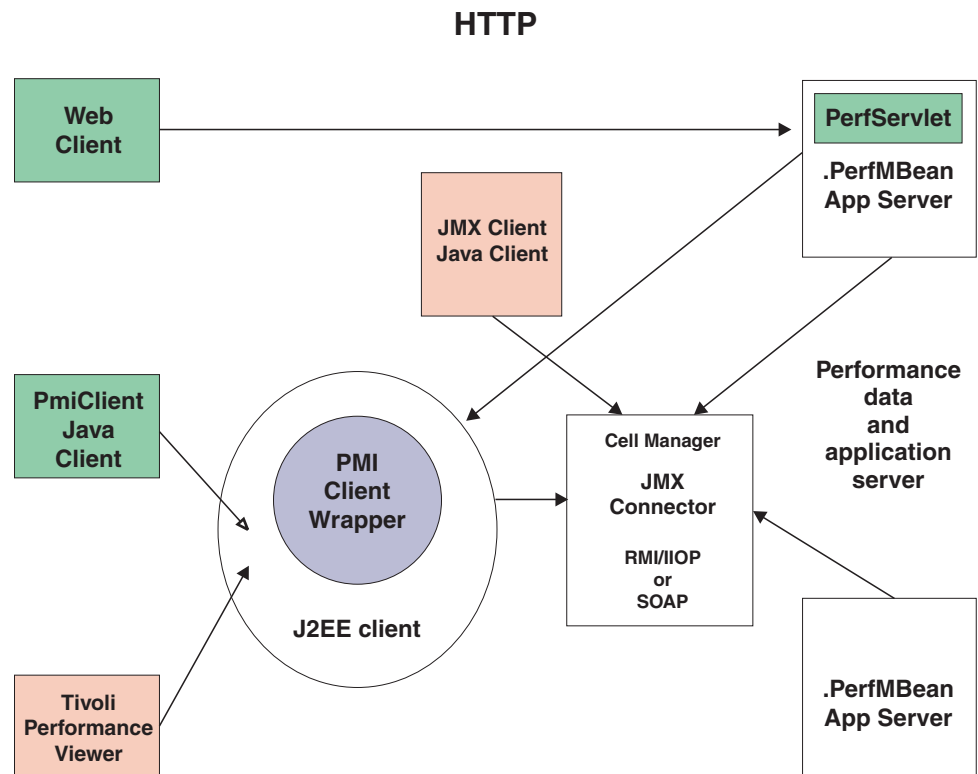
Each time a client retrieves performance data from a server, the data is returned in a subset of this structure; the form of the subset depends on the data retrieved. You can update the entire structure with new data, or update only part of the tree, as needed.

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

The following are additional Performance Monitoring Infrastructure (PMI)
interfaces:

- BoundaryStatistic
- BoundedRangeStatistic
- CountStatistic
- MBeanStatDescriptor
- MBeanLevelSpec
- New Methods in PmiClient
- RangeStatistic
- Stats
- Statistic
- TimeStatistic

The following PMI interfaces introduced in Version 4.0 are also supported:

- CpdCollection
- CpdData
- CpdEventListener and CpdEvent
- CpdFamily class
- CpdValue
  - CpdLong
  - CpdStat
  - CpdLoad
- PerfDescriptor
- PmiClient class

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

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

Using Version 5.0 PMI API in Version 3.5.5+ and Version 4.0.x
For Version 3.5.5+, follow these instructions:

Steps for this task

1. Make configuration changes.

   For PMI to interact correctly with Version 3.5.x application servers, you must
   upgrade both the Version 3.5.x run time environment and the PMI JAR files to
   the levels specified below. In addition, you must prepend the `repository.jar`,
   `ejs.jar`, and `ujc.jar` files from the upgraded Version 3.5.x run time
   environment to the PMI client's run time classpath.

   a. Change the Version 3.5.x run time environment.

      Ensure the Version 3.5 environment is Version 3.5.5 or later.

b. Change the PMI client's run time or development environment.

   Both the Version 5.0 PMI client and the Version 4.02 client can work with the Version 3.5.5+ WebSphere Application Server.

   Copy the `repository.jar`, `ujc.jar` and `ejs.jar` files from the `<WebSphere_35_installation_root>/lib directory` to each machine from which a PMI client is run.

   Prepend the Version 3.5.5+ `repository.jar`, `ujc.jar` and `ejs.jar` files to the PMI client's run time classpath.

2. Copy the XML configuration files from Version 4.0.2+.

   a. Get the `perf.jar` file from Version 4.0.

   b. Append the `perf.jar` file to the classpath of the Version 5.0 PMI client.

   Note: Ensure the Version 5.0 `pmi.jar` file and `pmiclient.jar` files come **before** the Version 4.0 `perf.jar` file.

3. Make programmatic changes.

   A new constructor for PmiClient allows a client to monitor Version 3.5.5 or later application servers. The new constructor takes three string parameters: hostName, serverName, and version.

   ```
   public PmiClient(String host, String port, String version)
   ```

   Using this constructor with "EPM" as the third parameter creates a PmiClient that can retrieve data from Version 3.5.5+ application servers.

   ```
   PmiClient pmiClnt = new PmiClient(hostName, portNumber, "EPM")
   ```

What to do next

Use Version 4.0 PmiClient API to write your own client application with WebSphere Application Server Version 4.0 and 3.5.5+. See the example code for using Version 4.0 API in Example: Performance Monitoring Infrastructure client (Version 4.0).

To run a Version 5.0 PMI client with a Version 4.0 server, the instructions are similar, except in substep 2 of step 1, you need to copy the `repository.jar` and `ujc.jar` files from a WebSphere Application Server, Version 4.0, installation.

Example: Performance Monitoring Infrastructure client (Version 4.0)

The following is a list of example Performance Monitoring Infrastructure (PMI) client code from Version 4.0:

```
/**
 * This is a sample code to show how to use PmiClient to collect PMI data.
 * You will need to use adminconsole to set instrumentation level (a level other
 * than NONE) first.
 *
 * <p>
 * End-to-end code path in 4.0:
 *     PmiTester -> PmiClient -> AdminServer -> appServer
 */


package com.ibm.websphere.pmi;

import com.ibm.websphere.pmi.*;
import com.ibm.websphere.pmi.server.*;
import com.ibm.websphere.pmi.client.*;
import com.ibm.ws.pmi.server.*;
import com.ibm.ws.pmi.perfServer.*;
```

```
import com.ibm.ws.pmi.server.modules.*;
import com.ibm.ws.pmi.wire.*;
import java.util.ArrayList;


/**
 * Sample code to use PmiClient API (old API in 4.0) and get CpdData/CpdCollection objects.
 *
 */
public class PmiTester implements PmiConstants {

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

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

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

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

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

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

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

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

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

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

```
    }

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

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

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

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

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

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

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

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

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

            // Or call processCpdCollection to get each data
            processCpdCollection(cpdCols[i], "");
        }

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

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

```
                        continue;
                    }

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

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

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

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

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

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

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

```
                        double value = cpdDouble.getValue();
                        System.out.println(myindent + "CpdData id=" + dataList[i].getId()
                                            + " type=double value=" + value);
                    }
                else if(cpdVal.getType() == TYPE_INT) {
                        CpdValue cpdInt = (CpdValue)cpdVal;
                        int value = (int)cpdInt.getValue();
                        System.out.println(myindent + "CpdData id=" + dataList[i].getId()
                                            + " type=int value=" + value);
                    }
            }

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

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

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

}
```

Example: Performance Monitoring Infrastructure client with new data structure

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

```
package 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
            nodeName = pmiClnt.getConnectedHost();
            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);
            }
```

```java
// 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;
        }

    }

    /**
```

```java
     * 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;
                }

        }

}
```

Example: Administering Java Management Extension-based interface

Examples

The following is example code directly using Java Management Extension (JMX) API:

```
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 2809
 * @param args[2] connectorType, optional, default is RMI 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     = "2809";
        String  connector = "RMI";
        if(args.length < 1) {
            System.err.println("ERROR: Usage: PmiJmxTest <host>
                                [<port>] [<connector>]");
            System.exit(2);
        }
        else {
            host = args[0];

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

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

        if(host == null) {
            host = "localhost";
        }
        if(port == null) {
            port = "2809";
        }
        if (connector == null) {
            connector = AdminClient.CONNECTOR_TYPE_RMI;
        }
        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() + ",

        // 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 {
```

```java
        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(findConfig(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(findConfig(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 + "    ");
    }
}

/**
 * Get PmiModuleConfig based on MBean ObjectName
 */
public PmiModuleConfig findConfig(ObjectName on) {
    if (on == null) return null;

    String type = on.getKeyProperty("type");
    System.out.println("findConfig: mbean type =" + type);

    for (int i=0; i<configs.length ; i++) {

        if (configs[i].getMbeanType().equals(type))
            return configs[i];
    }
    System.out.println("Error: cannot find the config");
```

```
                return null;

            }

            /**
             * Get PmiModuleConfig based on PMI module name */
            public PmiModuleConfig findConfig(String moduleName) {
                if (moduleName == null) return null;

                for (int i=0; i<configs.length ; i++) {

                    if (configs[i].getShortName().equals(moduleName))
                        return configs[i];
                }
                System.out.println("Error: cannot find the config");
                return null;

            }

            /**
             * 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 = findConfig(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++) {
            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(findConfig(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 your own monitoring applications with Performance Monitoring Infrastructure servlet

Before you begin

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 .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:

Steps for this task
1. Deploy the servlet on a single application server instance within the domain.
2. After the servlet deploys, you can invoke it to retrieve performance data for the entire domain.
   Invoke the performance servlet by accessing the following default URL:
   http://*hostname/wasPerfTool/servlet/perfservlet*

Results

The performance servlet provides performance data output as an XML document, as described by the provided document type definition (DTD). The output structure provided is called leaves. The paths that lead to the leaves provide the context of the data. See 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="pmi3"/>
</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/perfservletrefreshConfig=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/perfservletNode=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/perfservletServer=TradeApp
  ```

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

  ```
  http://hostname/wasPerfTool/servlet/perfservletNode=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/perfservletModule=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/perfservletModule=beanModule
```

To limit the data collection to the beanModule on the server TradeApp on the node rjones, invoke the following URL (split for publication):

```
http://hostname/wasPerfTool/servlet/perfservletNode=rjones
    &Server=TradeApp&Module=beanModule>
```

Accessing Performance Monitoring Infrastructure data through the Java Management Extension interface

Before you begin

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 basic getStats method as specified in JSR-077, while PerfMBean provides WebSphere Application Server PMI extensions for more flexibility and administration.

See "Developing an administrative client program" (not in this document) 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:

Steps for this task

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 invoke the no-arg operation getStats() on the MBean.

Developing Performance Monitoring Infrastructure interfaces (Version 4.0)

Before you begin

The Version 4.0 APIs are supported in this release, however, some data hierarchy changes have occurred in the PMI modules, including the enterprise bean and HTTP sessions modules. 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.

The getDataName and getDataId methods in PmiClient have also changed. They are now 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.

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

Steps for this task
1. Retrieve an initial collection or snapshot of performance data from the server.

   A client uses the CpdCollection interface to retrieve an initial collection or snapshot from the server. This snapshot, which is called Snapshot in this example, is provided in a hierarchical structure as described in data organization and hierarchy, and contains the current values of all performance data collected by the server. The snapshot maintains the same structure throughout the lifetime of the CpdCollection instance.
2. Process and display the data as specified.

   The client processes and displays the data as specified. Processing and display objects, for example, filters and GUIs, can register as CpdEvent listeners to data of interest. The listener works only within the same Java virtual machine (JVM). When the client receives updated data, all listeners are notified.
3. Display the new CpdCollection instance through the hierarchy.

   When the client receives new or changed data, the client can simply display the new CpdCollection instance through its hierarchy. When it is necessary to update the Snapshot collection, the client can use the update method to update Snapshot with the new data.

   ```
   Snapshot.update(S1);
   // ...later...
   Snapshot.update(S2);
   ```

Results

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

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).

See Performance: Resources for learning for a link to IBM business partners providing monitoring solutions for WebSphere Application Server.

Measuring data requests (Performance Monitoring Infrastructure Request Metrics)

Performance Monitoring Infrastructure (PMI) Request Metrics collects data by timing requests as they travel through WebSphere Application Server components. This data helps to identify run time and application problems. PMI Request Metrics logs the time spent at major points, such as the Web container, enterprise

bean container and database. These points are recorded in logs and can be written to Application Response Measurement (ARM) agents used by Tivoli monitoring tools.

If you plan to run in a production environment, plan to filter by IP address - a specific IP address using a synthetic transaction generator. If you choose to enable request metrics, but not filter by a specific IP address, performance can be impacted significantly.

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

Performance Monitoring Infrastructure Request Metrics

Performance Monitoring Infrastructure (PMI) Request Metrics helps identify run time and application performance problems by capturing process hop response times in multi-tiered applications and recording the data in system logs.

For requests that start from either an HTTP or enterprise bean remote requests, Request Metrics captures response times for the initiating request and any related downstream enterprise bean invocations and Java Database Connectivity (JDBC) calls. Request Metrics also provides the same information on process hop response time through the Application Response Measurement (ARM) interface.

When active, Request Metrics compares each incoming request to a set of known filters. If the request matches any filter with a trace level greater than TRACE_NONE, trace records are generated for that request.

Typically, requests enter the system and create processes that fan out across several nodes within a distributed system. Each process can further fan out and call other processes. When the processes fan out, trace records are generated for each process. Then, these trace records can be correlated together to build a sequence diagram of the response times for the request. The processes are only recorded if they are generated through a remote enterprise bean call.

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, but can be used with agents conforming to the ARM 2.0 standard. Contact your ARM agent provider for information on whether their ARM agent is supported with WebSphere Application Server.

See Performance: Resources for learning 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 StdOut and can be used for real-time and historical analysis.

Incoming HTTP requests

For HTTP requests arriving at a WebSphere Application Server it is possible to filter on the URI and client IP address.

- **Client 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 as 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 client IP address filters.
- **Filter combinations.** If both URI and Client 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, you can use the asterisk (*) to provide a mask. The asterisk must always be the last character of a filter pattern.

Performance Monitoring Infrastructure Request Metrics data output

The trace record format for Performance Monitoring Infrastructure (PMI) Request Metrics data output follows:

```
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
```

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 server logs. 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.

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.

The information following the parent and current correlators describes the trace event. The elapsed field shows the elapsed time for the operation described in the type and detail fields. The elapsed time includes the time for all operations having parent correlators equal to the current correlator of the trace record being examined.

The type and detail fields are described as follows:

- **Universal Resource Identifier (URI):** The trace record was generated by a Web component. The URI is the name of the URI used to invoke the request.
- **Enterprise bean:** The fully qualified package and method name of the enterprise bean
- **Java Database Connectivity (JDBC):** The values select, update, insert or delete for prepared statements. For non-prepared statements, the full statement can appear.

Configuring Request Metrics

Before you begin

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 **Problem Determination** > **PMI Request Metrics** from the administrative console navigation tree.

Tasks included in configuring Request Metrics:

Steps for this task
1. Enable Request Metrics.
2. (Optional) Enable Application Response Measurement (ARM).
3. (Optional) Enable Request Metrics filters.
4. Add and remove Request Metrics filters.
5. Set the trace level in Request Metrics.

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.

Steps for this task

1. Open the administrative console.
2. Click **Problem Determination** > **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**.

Enabling Application Response Measurement

Before you begin

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 Request Metrics is supported by the ARM agent implementation. ARM support is dependent on Request Metrics support.

You can learn more about ARM agents in Performance: Resources for Learning.

Steps for this task

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 **Problem Determination** > **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**.

Enabling Performance Monitoring Infrastructure Request Metrics filters

Performance Monitoring Infrastructure (PMI) Request Metrics compares each incoming request to a set of known filters, but you need to enable these filters.

Steps for this task

1. Open the administrative console.
2. Click **Problem Determination** > **PMI Request Metrics** in the administrative console navigation tree.
3. Click **filters**.
4. Click *filter type*.
5. Select the check box in the **enable** field under the Configuration tab.
6. Click **Apply** or **OK**.
7. Click **Save**.

You can enable or disable a filter group. If the group is enabled, you can enable or disable individual filters.

Adding and removing Performance Monitoring Infrastructure Request Metrics: filters

To add or remove Performance Monitoring Infrastructure (PMI) Request Metrics filters:

Steps for this task
1. Open the administrative console.
2. Click **Problem Determination** > **PMI Request Metrics** in the console navigation tree.
3. Click **filters**.
4. Click **New**.
5. Choose a filter type from the drop down box in the **type** field under the Configuration tab.
6. (Optional) Select the check box in the **enable** field to enable the filter.
7. Click **Apply** or **OK**.
8. Click **Save**.

   Individual filters are composed of an indicator and an IP address. Use the indicator to determine whether the individual filter is active. The IP address is composed of a standard dotted IP address.

Setting the trace level in Performance Monitoring Infrastructure Request Metrics

To set the trace level to generate records:

Steps for this task
1. Open the administrative console.
2. Click **Problem Determination** > **PMI Request Metrics** in the administrative console navigation tree.
3. Find **traceLevel** in the Configuration tab.
4. Select the desired trace level from the drop down list box.

   To set the Request Metrics trace level to generate records, make sure the trace level is set to a value greater than NONE.
5. Click **Apply** or **OK**.
6. Click **Save**.

Performance Monitoring Infrastructure Request Metrics configuration settings

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 **Problem Determination** > **PMI Request Metrics**.

enable: Enables PMI Request Metrics.

When disabled, the Request Metrics function is disabled.

enableARM: 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.

traceLevel: 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 - enables additional information over hops, but is not as performance intensive as DEBUG; DEBUG - full detailed trace.

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

To view this administrative console page, click **Problem Determination** > **PMI Request Metrics** > **filters**.

type: Specifies the type of request metrics filter.

enable: Specifies whether this filter is enabled.

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 **Problem Determination** > **PMI Request Metrics** > **filters** > *filter*.

type: Specifies the type of Request Metrics filter.

enable: Specifies whether this filter is enabled.

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

To view this administrative console page, click **Problem Determination** > **PMI Request Metrics** > **filters** > *filter* > **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: Specifies whether a filter value is enabled.

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

To view this administrative console page, click **Problem Determination** > **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: Generating trace records from PMI Request Metrics

Examples

Use HitCount enterprise bean `/webapp/examples/hitcount?source=EJB` where the servlet is deployed on one machine - 192.168.0.1, and the enterprise bean `Increment.jar` file is deployed on a second machine - 192.168.0.2.

In this example, both machines are used as clients.

To illustrate the use of client IP filtering, one client IP filter (192.168.0.2) is defined and enabled. This action allows tracing of requests originating from the enterprise bean machine through `http://192.168.0.1/webapp/examples/hitcount?source=EJB`. However, requests originating from this machine are not traced since the client IP address is not in the filter list.

By only creating a client IP filter, any request from that client IP address is effectively traced. This tool can be effective for locating performance problems with systems under load. If the normal load is originating from other IP addresses, then their requests are not traced. By using the defined client IP address to generate requests, you can see performance bottlenecks at the various hops by comparing the trace records of the loaded system to trace records from a non-loaded run. This ability can help focus tuning efforts to the correct node and process within a complex deployment environment.

Make sure Request Metrics is enabled using the administrative console. Also, make sure the trace level is set to at least Tricepses. Using the configuration listed above, send a request through the HitCount servlet from the enterprise bean machine `http://192.168.0.1/webapp/examples/hitcount?source=EJB`.

In this example, at least two trace records are generated:
- A trace record for the servlet execution will appear on 192.168.0.1
- A trace record for the increment bean method invocation will appear on 192.168.0.2

The trace record appearing on 192.168.0.1 should appear very similar to the following:
```
PMRM0003I: parent:ver=1,ip=192.168.0.1,time=1016556186102,pid=884,reqid=40,event=0
           - current:ver=1,ip=192.168.0.1,time=1016556186102,pid=884,reqid=40,event=1
type=URI detail=/webapp/examples/hitcount elapsed=60
```

The trace record appearing on 192.168.0.2 should appear very similar to the following:
```
PMRM0003I: parent:ver=1,ip=192.168.0.1,time=1016556186102,pid=884,reqid=40,event=1
           - current:ver=1,ip=192.168.0.2,time=1016556122505,pid=9321,reqid=40,event=1
type=EJB detail=com.ibm.websphere.examples.Inc.IncBean.increment elapsed=40
```

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:
- Performance Monitoring Infrastructure (PMI) Request Metrics
- Monitoring performance with third-party tools

Performance Monitoring Infrastructure (PMI) Request Metrics

- **Systems Management: Application Response Measurement (ARM)**
  (http://www.opengroup.org/publications/catalog/c807.htm)

  The Open Group ARM specifications.

Monitoring performance with third-party tools

- **Enterprise Web Application Management** (http://www-
  3.ibm.com/software/webservers/pw/dhtml/wsperformance/performance_bpsolutions.html)
  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.

Chapter 3. Tuning performance

Symptom table

Take a shortcut into tuning by reviewing the symptom table. The table is designed for easy access to symptoms and a quick link to tuning information related to that symptom. The table contains the following types of information:

- **The symptom**
  The left column lists symptoms and descriptions. The symptoms can be specific or general.
- **Additional information**
  The right column lists a link that leads to additional information about the symptom.

Symptom	Additional information
Throughput and response time are undesirable.	Processor speed
AIX: Memory allocation error Solaris: Too many files open	AIX file descriptors (ulimit) or Solaris file descriptors (ulimit)
Solaris: The server stalls during peak periods, responses take minutes, processor utilization remains high with all activity in the system processes, and netstat shows many sockets are open to port 80 in CLOSE_WAIT or FIN_WAIT_2 state.	Solaris tcp_time_wait_interval and Solaris tcp_fin_wait_2_flush_interval
Windows NT or 2000: Netstat shows too many sockets are in TIME_WAIT.	Windows NT or 2000 TcpTimedWaitDelay
Throughput is undesirable and the application server priority has not been adjusted.	Adjusting the operating system priority of the WebSphere Application Server process
Under load, client requests do not arrive at the Web server because they time out or are rejected.	For IBM HTTP Server on Windows NT, see ListenBackLog
Windows NT or 2000: WebSphere Application Server performance decreased after an application server from another vendor was installed.	Microsoft Internet Information Server (IIS) properties
Tivoli Performance Viewer Percent Maxed Metric indicates that the Web container thread pool is too small.	Thread pool
Netstat shows too many TIME_WAIT state sockets for port 9080.	MaxKeepAliveConnections, MaxKeepAliveRequests
Too much disk input/output occurs due to paging.	Heap size settings

Tivoli Performance Viewer's Percent Used Metric for a data source connection pool indicates the pool size is too large.	Connection pool size
Tivoli Performance Viewer's Prepared Statement Cache Discards Metric indicates the data source prepared statement cache size is too small.	Prepared statement cache size
Too much disk input/output occurs due to DB2 writing log records.	DB2 MinCommit
Tivoli Performance Viewer Percent Maxed Metric indicates the Object Request Broker thread pool is too small.	Queuing and enterprise beans
Tivoli Performance Viewer Java Virtual Machine Profiler Interface (JVMPI) indicates over-utilization of objects when too much time is being spent in garbage collection.	Detecting over-utilization of objects
Tivoli Performance Viewer Used Memory Metric shows memory leaks and Java displays an `Out of Memory` exception.	Detecting memory leaks
Throughput, response time and scalability are undesirable.	If the application permits, exploit dynamic fragment caching

Tuning basics

This tuning guide describes tuning improvements for WebSphere Application Server through general recommendations and a description of specific tuning methodologies. Hints and tips on the various factors and variables in the tuning guide can help to enhance performance.

Use the tuning guide, along with its examples and resources, to expand your tuning experience. Tuning is an ongoing learning process. Results can vary depending on the configuration and application, and thus, might differ from those in presented in this guide.

What influences tuning?

The following are are parameters that can affect the performance of WebSphere Application Server:

- The application being used
- Hardware capacity and settings
- Operating system settings
- Web server
- WebSphere application server process
- Java Virtual Machine (JVM)
- Database

Each parameter has its own tuning options, varying in importance and impact. Each parameter is explained in detail in the Individual Performance Parameters section of this document.

For convenience, this document also describes procedures for setting parameters in other products. Because these products can change, consider these descriptions as suggestions.

Types of tuning

The two types of tuning are application tuning and parameter tuning.

Although application tuning sometimes offers the greatest tuning improvements, this document focuses on tuning individual performance parameters and the interactions between them.

The (WebSphere Application Server Development Best Practices for Performance and Scalability) whitepaper, addresses application tuning by describing development best practices for Web applications containing servlets, Java Server Pages (JSP) files, Java Database Connectivity (JDBC), and enterprise applications containing enterprise bean components.

Parameter tuning

Parameter tuning is the art of changing WebSphere Application Server settings with the goal of improving performance. The values suggested in this document are general guidelines. The optimal settings for your environment can vary significantly. In addition, remember that after tuning one bottleneck away, you can encounter another, unrelated bottleneck. If so, you might not experience the performance improvement until both bottlenecks have been removed.

This section discusses two kinds of tuning parameters:
- Tuning parameters with high performance results
- Tuning parameters for avoiding failure

Tuning parameters with high performance results: These parameters are a subset of all other parameters and have an important effect on performance. Because these parameters are application-dependent, the appropriate settings for the application and environment might be different.

The following table lists various high performance-enhancing tuning parameters:

Adjusting WebSphere Application Server system queues
Application assembly performance checklist
Using pass-by-value versus pass-by-reference (NoLocalCopies)
Adjusting Solaris TCP parameters
Tuning Java memory
Adjusting MaxRequestsPerChild: on Linux with IBM HTTP Server
Adjusting connection pool size
Adjusting prepared statement cache size
Web server configuration reload interval

Tuning parameters for avoiding failures: Tuning the following parameters can help to prevent functional problems:

Number of connections to DB2: To establish more connections than DB2 sets up by default
Allow thread allocation beyond maximum has been selected and the system is overloaded because too many threads are allocated.
Using TCP Sockets for DB2 on Linux: For local databases
Connection pool size: Ensure enough connections for transaction processing with Entity EJBs and for avoiding deadlock.

Adjusting the queues in WebSphere Application Server

WebSphere Application Server has a series of interrelated components that must be harmoniously tuned to support the custom needs of your end-to-end e-business application. These adjustments help the system achieve maximum throughput while maintaining the overall stability of the system.

Queuing network

WebSphere Application Server establishes a queuing network, which is a group of interconnected queues that represent various components. These queues include the network, Web server, Web container, EJB container, data source, and possibly a connection manager to a custom back-end system. Each of these resources represents a queue of requests waiting to use that resource.
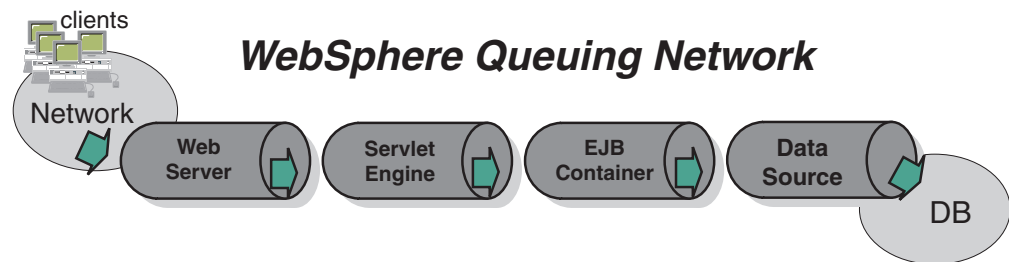
The WebSphere queues are load-dependent resources. The average service time of a request depends on the number of concurrent clients.

Closed queues: Most of the queues that make up the queuing network are closed queues. A closed queue places a limit on the maximum number of requests present in the queue, while an open queue has no limit.

A closed queue allows for tight management of system resources. For example, the Web container's thread pool setting controls the size of the Web container queue. If the average servlet running in a Web container creates 10MB of objects during each request, then a value of 100 for thread pools would limit the memory consumed by the Web container to 1GB.

In a closed queue, requests can be either active or waiting. An active request is either doing work or waiting for a response from a downstream queue. For example, an active request in the Web server is either doing work (such as retrieving static HTML) or waiting for a request to complete in the Web container. A waiting request is waiting to become active. The request remains in the waiting state until one of the active requests leaves the queue.

All Web servers supported by WebSphere Application Server are closed queues, as are WebSphere Application Server data sources. Web containers can be configured as either open or closed queues. In general, it is best to make them closed queues.

EJB containers are open queues; if there are no threads available in the pool, a new one will be created for the duration of the request.

If enterprise beans are being called by servlets, the Web container limits the number of total concurrent requests into an EJB container, because the Web container also has a limit. This is true only if enterprise beans are called from the servlet thread of execution. Nothing prevents you from creating threads and bombarding the EJB container with requests. Thus, servlets should not create their own work threads.

Queue settings in WebSphere Application Server: The following outlines the various queue settings:
- IBM HTTP Server: MaxClients (for UNIX) and ThreadsPerChild (for Windows NT or 2000)
- Web container
  - Thread pool
  - MaxKeepAliveConnections
  - MaxKeepAliveRequests
- Object Request Broker thread pool size
- Connection pool size
- Prepared statement cache size

Determining the settings

The following section outlines a methodology for configuring the WebSphere Application Server queues. The dynamics of an individual system can be dramatically changed by moving resources, for example, moving the database server onto another machine, or providing more powerful resources, for example a faster set of CPUs with more memory. Thus, you can adjust the tuning parameters to a specific configuration of the production environment.

Queuing before WebSphere: The first rule of tuning is to minimize the number of requests in WebSphere Application Server queues. In general, requests should wait in the network (in front of the Web server), rather than waiting in WebSphere Application Server. This configuration allows only those requests that are ready to be processed to enter the queuing network. To accomplish this, specify that the queues furthest upstream (closest to the client) are slightly larger, and that the queues further downstream (furthest from the client) are progressively smaller.
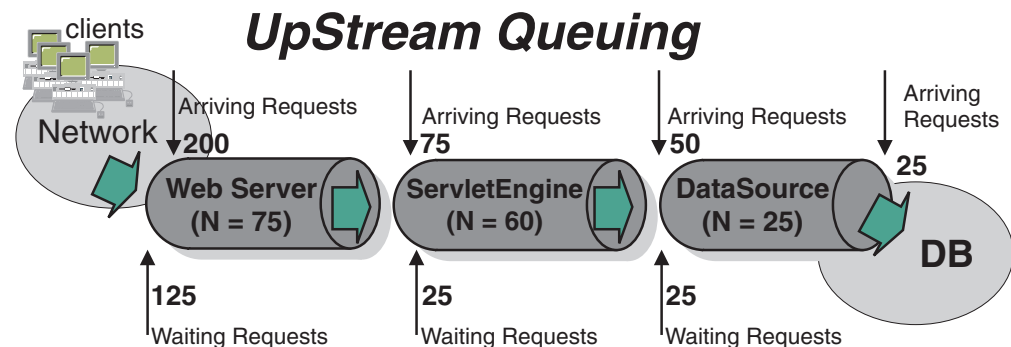
The queues in this example queuing network become progressively smaller as work flows downstream. When 200 client requests arrive at the Web server, 125 requests remain queued in the network because the Web server is set to handle 75 concurrent clients. As the 75 requests pass from the Web server to the Web

container, 25 remain queued in the Web server and the remaining 50 are handled by the Web container. This process progresses through the data source until 25 user requests arrive at the final destination, the database server. Because there is work waiting to enter a component at each point upstream, no component in this system must wait for work to arrive. The bulk of the requests wait in the network, outside of WebSphere Application Server. This type of configuration adds stability, because no component is overloaded. The Edge Server can be used to direct waiting users to other servers in a WebSphere Application Server cluster.

Drawing a throughput curve

You can use a test case that represents the full spirit of the production application by either exercising all meaningful code paths or using the production application. Run a set of experiments to determine when the system capabilities are fully stressed (the **saturation point**). Conduct these tests after most of the bottlenecks have been removed from the application. The typical goal of these tests is to drive CPUs to near 100% utilization.

Start the initial baseline experiment with large queues. This allows maximum concurrency through the system. For example, start the first experiment with a queue size of 100 at each of the servers in the queuing network: Web server, Web container and data source.

Next, begin a series of experiments to plot a throughput curve, increasing the concurrent user load after each experiment. For example, perform experiments with 1 user, 2 users, 5, 10, 25, 50, 100, 150 and 200 users. After each run, record the throughput (requests per second) and response times (seconds per request).

The curve resulting from the baseline experiments should resemble the typical throughput curve shown as follows:

Throughput curve
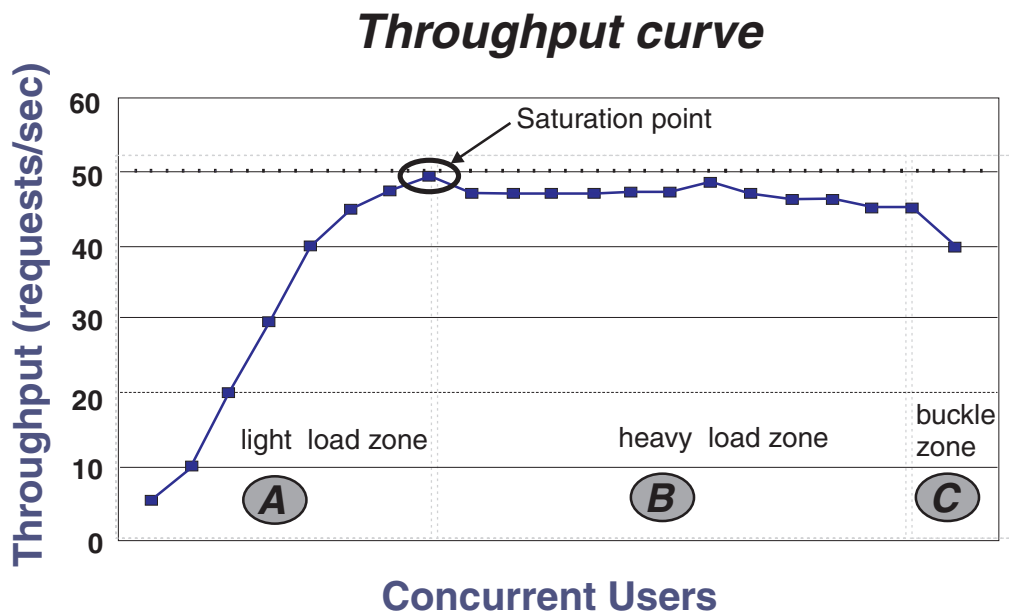
The throughput of WebSphere Application Server is a function of the number of concurrent requests present in the total system. Section A, the light load zone, shows that as the number of concurrent user requests increases, the throughput increases almost linearly with the number of requests. This reflects that, at light loads, concurrent requests face very little congestion within the WebSphere

Application Server system queues. At some point, congestion starts to develop and throughput increases at a much lower rate until it reaches a saturation point that represents the maximum throughput value, as determined by some bottleneck in the WebSphere Application Server system. The most manageable type of bottleneck occurs when the CPUs of the WebSphere Application Server machines become fully utilized. This is desirable because a CPU bottleneck can be fixed by adding additional or more powerful CPUs.

In the heavy load zone or Section B, as the concurrent client load increases, throughput remains relatively constant. However, the response time increases proportionally to the user load. That is, if the user load is doubled in the heavy load zone, the response time doubles. At some point, represented by Section C, the buckle zone, one of the system components becomes exhausted. At this point, throughput starts to degrade. For example, the system might enter the buckle zone when the network connections at the Web server exhaust the limits of the network adapter or if the requests exceed operating system limits for file handles.

If the saturation point is reached by driving CPU utilization close to 100%, you can move on to the next step. If the saturation CPU occurs before system utilization reaches 100%, there is likely another bottleneck that is being aggravated by the application. For example, the application might be creating Java objects causing excessive garbage collection bottlenecks in Java.

There are two ways to manage application bottlenecks: remove the bottleneck or clone the bottleneck. The best way to manage a bottleneck is to remove it. You can use a Java-based application profiler, such as WebSphere Studio Application Developer (WSAD), Performance Trace Data Visualizer(PTDV), Optimizeit, JProbe or Jinsight to examine overall object utilization.

Queue adjustments: The number of concurrent users at the throughput saturation point represents the maximum concurrency of the application. For example, if the application saturated WebSphere Application Server at 50 users, 48 users might give best combination of throughput and response time. This value is called the Max Application Concurrency value. Max Application Concurrency becomes the preferred value for adjusting the WebSphere Application Server system queues. Remember, it is desirable for most users to wait in the network; therefore, queue sizes should increase when moving downstream farther from the client. For example, given a Max Application Concurrency value of 48, start with system queues at the following values: Web server 75, Web container 50, data source 45. Perform a set of additional experiments adjusting these values slightly higher and lower to find the best settings.

The Tivoli Performance Viewer can be used to determine the number of concurrent users through the Servlet Engine Thread Pool Concurrently Active Threads metric.

In IBM performance experiments, throughput has increased by 10-15% when the Web container transport maximum keep-alive are adjusted to match the maximum number of Web container threads.

Adjusting queue settings for access patterns: In many cases, only a fraction of the requests passing through one queue enters the next queue downstream. In a site with many static pages, many requests are fulfilled at the Web server and are not passed to the Web container. In this circumstance, the Web server queue can be significantly larger than the Web container queue. In the previous section, the Web

server queue was set to 75 rather than closer to the value of Max Application Concurrency. Similar adjustments need to be made when different components have different execution times.
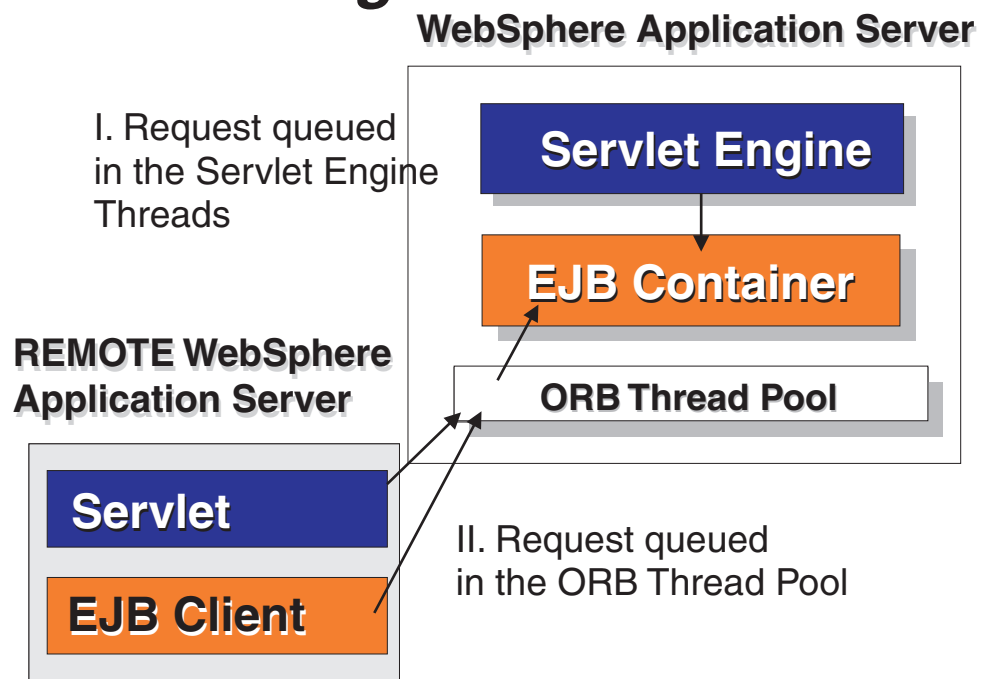
For example, in an application that spends 90% of its time in a complex servlet and only 10% making a short JDBC query, on average 10% of the servlets are using database connections at any time, so the database connection queue can be significantly smaller than the Web container queue. Conversely, if much of a servlet execution time is spent making a complex query to a database, consider increasing the queue values at both the Web container and the data source. Always monitor the CPU and memory utilization for both the WebSphere Application Server and the database servers to ensure the CPU or memory are not being saturated.

Queuing and enterprise beans

Method invocations to enterprise beans are only queued for remote clients, making the method call. An example of a remote client is an EJB client running in a separate Java Virtual Machine (another address space) from the enterprise bean. In contrast, no queuing occurs if the EJB client (either a servlet or another enterprise bean) is installed in the same JVM that the EJB method runs on and the same thread of execution as the EJB client.

Remote enterprise beans communicate by using the RMI/IIOP protocol. Method invocations initiated over RMI/IIOP are processed by a server-side ORB. The thread pool acts as a queue for incoming requests. However, if a remote method request is issued and there are no more available threads in the thread pool, a new thread is created. After the method request completes the thread is destroyed. Therefore, when the ORB is used to process remote method requests, the EJB container is an open queue, due to the use of unbounded threads. The following illustration depicts the two queuing options of enterprise beans.
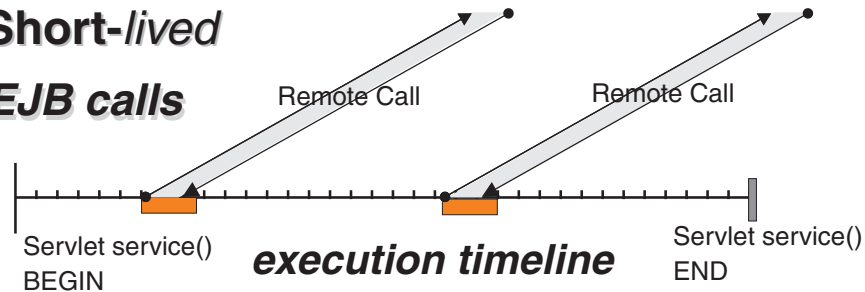
EJB Queuing

WebSphere Application Server

I. Request queued in the Servlet Engine Threads

Servlet Engine

EJB Container

REMOTE WebSphere Application Server

ORB Thread Pool

Servlet

II. Request queued in the ORB Thread Pool

EJB Client

When configuring the thread pool, it is important to understand the calling patterns of the EJB client. If a servlet is making a small number of calls to remote enterprise beans and each method call is relatively quick, consider setting the number of threads in the ORB thread pool to a value lower than the Web container thread pool size value.
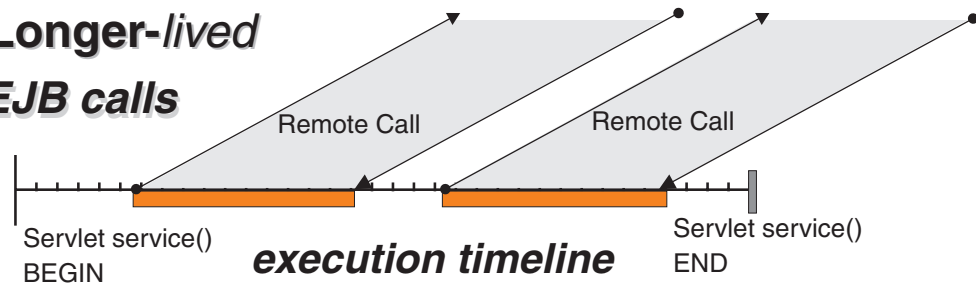
Short-_lived_

EJB calls

Remote Call   Remote Call

Servlet service()   **execution timeline**   Servlet service()
BEGIN   END

Longer-_lived_

EJB calls

Remote Call   Remote Call

Servlet service()   **execution timeline**   Servlet service()
BEGIN   END

Tivoli Performance Viewer shows a metric called percent maxed used to determine how much of the time all of the configured threads are in use. If this value is consistently in the double-digits, then the ORB could be a bottleneck and the number of threads should be increased.

The degree to which the ORB thread pool value needs to be increased is a function of the number of simultaneous servlets (that is, clients) calling enterprise beans and the duration of each method call. If the method calls are longer or the applications spend a lot of time in the ORB, consider making the ORB thread pool size equal to the Web container size. If the servlet makes only short-lived or quick calls to the ORB, servlets can potentially reuse the same ORB thread. In this case, the ORB thread pool can be small, perhaps even one-half of the thread pool size setting of the Web container.

Queuing and clustering

The capabilities for cloning application servers can be a valuable asset in configuring highly scalable production environments. This is especially true when the application is experiencing bottlenecks that are preventing full CPU utilization of Symmetric Multiprocessing (SMP) servers. When adjusting the WebSphere Application Server system queues in clustered configurations, remember that when a server is added to a cluster, the server downstream receives twice the load.

Clustering and Queuing
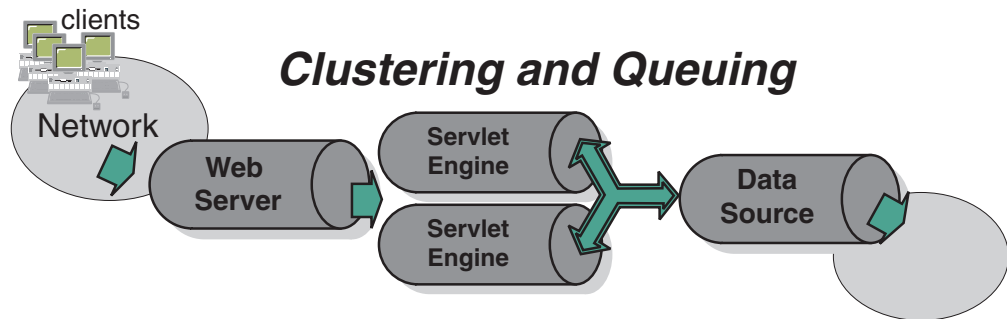
Two Web container clones are located between a Web server and a data source. It is assumed the Web server, servlet engines and data source (but not the database) are all running on a single SMP server. Given these constraints, the following queue considerations need to be made:

- Web server queue settings can be doubled to ensure ample work is distributed to each Web container.
- Web container thread pools can be reduced to avoid saturating a system resource such as CPU or another resource that the servlets are using.
- The data source can be reduced to avoid saturating the database server.
- Java heap parameters can be reduced for each instance of the application server. For versions of the JVM shipped with WebSphere Application Server, it is crucial that the heap from all JVMs remain in physical memory. Therefore, if a cluster of four JVMs are running on a system, enough physical memory must be available for all four heaps.

Tuning Secure Sockets Layer

The following are two types of Secure Socket Layer (SSL) performance:
- Handshake
- Bulk encryption/decryption

Overview of handshake and bulk encryption and decryption

When an SSL connection is established, an SSL handshake occurs. After a connection is made, SSL performs bulk encryption and decryption for each read-write. The performance cost of an SSL handshake is much larger than that of bulk encryption and decryption.

How to enhance SSL performance

In order to enhance SSL performance, the number of individual SSL connections and handshakes must be decreased.

Decreasing the number of connections increases performance for secure communication through SSL connections, as well as non-secure communication through simple TCP connections. One way to decrease individual SSL connections is to use a browser that supports HTTP 1.1. Decreasing individual SSL connections could be impossible for some users if they cannot upgrade to HTTP 1.1.

Another common approach is to decrease the number of connections (both TCP and SSL) between two WebSphere Application Server components. The following guidelines help to ensure the HTTP transport of the application server is configured so that the Web server plug-in does not repeatedly reopen new connections to the application server:

- The maximum number of keep-alives should be, at minimum, as large as the maximum number of requests per thread of the Web server (or maximum number of processes for IHS on UNIX). In other words, make sure the Web server plug-in is capable of obtaining a keep-alive connection for every possible concurrent connection to the application server. Otherwise, the application server will close the connection after a single request has been processed. Also, the maximum number of threads in the Web container thread pool should be larger than the maximum number of keep-alives, in order to prevent the Web container threads from being consumed with keep-alive connections.

- The maximum number of requests per keep-alive connection can also be increased. The default value is 100, which means the application server will close the connection from the plug-in after 100 requests. The plug-in would then have to open a new connection. The purpose of this parameter is to prevent denial of service attacks when connecting to the application server and continuously send requests in order to tie up threads in the application server.

- Use a hardware accelerator if the system performs several SSL handshakes.

  Hardware accelerators currently supported by WebSphere Application Server only increase the SSL handshake performance, not the bulk encryption/decryption. An accelerator typically only benefits the Web server because Web server connections are short-lived. All other SSL connections in WebSphere Application Server are long-lived.

- Use an alternative cipher suite with better performance.

  The performance of a cipher suite is different with software and hardware. Just because a cipher suite performs better in software does not mean a cipher suite will perform better with hardware. Some algorithms are typically inefficient in hardware (for example, DES and 3DES), however, specialized hardware can provide efficient implementations of these same algorithms.

  The performance of bulk encryption and decryption is affected by the cipher suite used for an individual SSL connection.The following chart displays the performance of each cipher suite. The test software calculating the data was IBM JSSE for both the client and server software, which used no crypto hardware support. The test did not include the time to establish a connection, but only the time to transmit data through an established connection. Therefore, the data reveals the relative SSL performance of various cipher suites for long running connections.

  Before establishing a connection, the client enabled a single cipher suite for each test case. After the connection was established, the client timed how long it took to write an integer to the server and for the server to write the specified number of bytes back to the client. Varying the amount of data had negligible effects on the relative performance of the cipher suites.
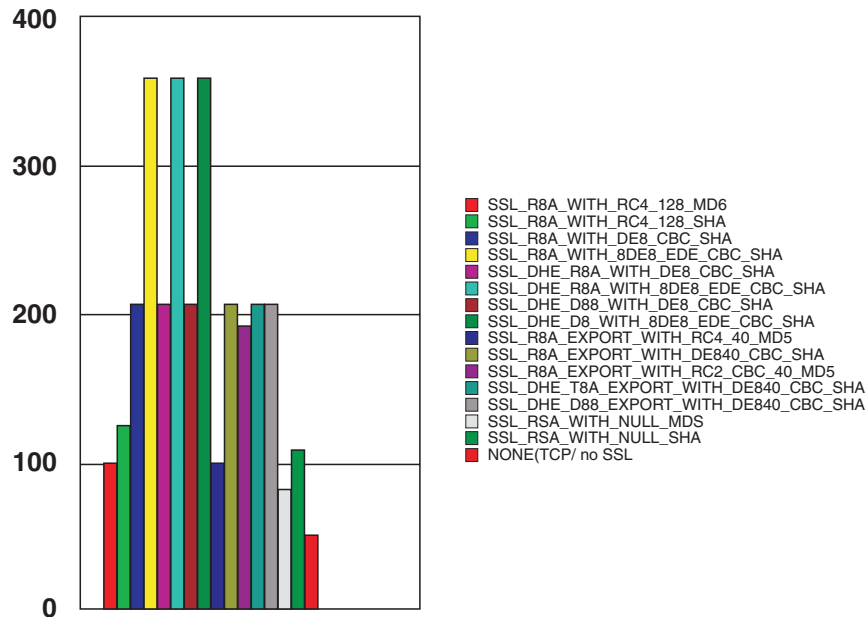
An analysis of the above data reveals the following:

– Bulk encryption performance is only affected by what follows the WITH in the cipher suite name. This is expected since the portion before the WITH identifies the algorithm used only during the SSL handshake.

– MD5 and SHA are the two hash algorithms used to provide data integrity. MD5 is 25% faster than SHA, however, SHA is more secure than MD5.

– DES and RC2 are slower than RC4. Triple DES is the most secure, but the performance cost is high when using only software.

– The cipher suite providing the best performance while still providing privacy is SSL_RSA_WITH_RC4_128_MD5. Even though SSL_RSA_EXPORT_WITH_RC4_40_MD5 is cryptographically weaker than RSA_WITH_RC4_128_MD5, the performance for bulk encryption is the same. Therefore, as long as the SSL connection is a long-running connection, the difference in the performance of high and medium security levels is negligible. It is recommended that a security level of high be used, instead of medium, for all components participating in communication only among WebSphere Application Server products. Make sure that the connections are long running connections.

Application assembly performance checklist Application assembly tools are used to assemble J2EE components and modules into J2EE applications. Generally, this 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 runtime performance of the deployed application. The most important parameters and advice for finding optimal settings follow:

– Enterprise bean modules

  - Entity EJBs - Bean cache

  - Method extensions - Isolation level

  - Method extensions - Access intent

  - Container transactions

– Web module

- Web application - Distributable
- Web application - Reload interval
- Web application - Reload enabled
- Web application - Web components - Load on startup

Enterprise bean modules

Note that although WebSphere Application Server 5.0 also supports EJB 2.0, the following information refers to EJB 1.1 settings.

Entity EJBs - Bean cache: WebSphere Application Server provides significant flexibility in the management of database data with Entity EJBs. The Entity EJBs **Activate At** and **Load At** configuration settings specify how and when to load and cache data from the corresponding database row data of an enterprise bean. These configuration settings provide the capability to specify enterprise bean commit options A, B or C, as specified in the EJB 1.1 specification.

Guide: The **Activate At** and **Load At** settings are detailed below along with specific settings to achieve each of the enterprise bean commit options A, B and C.

Commit option A provides maximum enterprise bean performance by caching database data outside of the transaction scope. Generally, commit option A is only applicable where the EJB container has exclusive access to the given database. Otherwise, data integrity is compromised. Commit option B provides more aggressive caching of entity EJB object instances, which can result in improved performance over commit option C, but also results in greater memory usage. Commit option C is the most common real-world configuration for Entity EJBs.

- **Bean cache - Activate At** This setting specifies the point at which an enterprise bean is activated and placed in the cache. Removal from the cache and passivation are also governed by this setting. Valid values are **Once** and **Transaction**. **Once** indicates that the bean is activated when it is first accessed in the server process, and passivated (and removed from the cache) at the discretion of the container, for example, when the cache becomes full. **Transaction** indicates that the bean is activated at the start of a transaction and passivated (and removed from the cache) at the end of the transaction. The default value is **Transaction**.

- **Bean cache - Load At** This setting specifies when the bean loads its state from the database. The value of this property implies whether the container has exclusive or shared access to the database. Valid values are **Activation** and **Transaction**. **Activation** indicates the bean is loaded when it is activated and implies that the container has exclusive access to the database. **Transaction** indicates that the bean is loaded at the start of a transaction and implies that the container has shared access to the database. The default is **Transaction**.

  The settings of the **Activate At** and **Load At** properties govern which commit options are used.

  – For commit option A (exclusive database access), use **Activate At = Once** and **Load At = Activation**. This option reduces database input/output by avoiding calls to the ejbLoad function, but serializes all transactions accessing the bean instance. Option A can increase memory usage by maintaining more objects in the cache, but can provide better response time if bean instances are not generally accessed concurrently by multiple transactions.

  – For commit option B (shared database access), use **Activate At = Once** and **Load At = Transaction**. Option B can increase memory usage by maintaining more objects in the cache. However, because each transaction creates its own copy of an object, there can be multiple copies of an instance in memory at

any given time (one per transaction), requiring the database be accessed at each transaction. If an enterprise bean contains a significant number of calls to the ejbActivate function, using option B can be beneficial because the required object is already in the cache. Otherwise, this option does not provide significant benefit over option A.

– For commit option C (shared database access), use **Activate At = Transaction** and **Load At = Transaction**. This option can reduce memory usage by maintaining fewer objects in the cache, however, there can be multiple copies of an instance in memory at any given time (one per transaction). This option can reduce transaction contention for enterprise bean instances that are accessed concurrently but not updated.

Method extensions - Isolation level: WebSphere Application Server enterprise bean method extensions provide settings to specify the level of transactional isolation used when accessing data. The valid values are:

- Serializable
- Repeatable Read
- Read Committed
- Read Uncommitted

Guide: Defined below, isolation level settings specify various degrees of runtime data integrity provided by the corresponding database. First, choose a setting that meets data integrity requirements for the given application and specific database characteristics.

Isolation level also plays an important role in performance. Higher isolation levels reduce performance by increasing row locking and database overhead while reducing data access concurrency. Various databases provide different behavior with respect to the isolation settings. In general, **Repeatable Read** is an appropriate setting for DB2 databases. **Read Committed** is generally used for Oracle. Oracle does not support **Repeatable Read** and will translate this setting to the highest isolation level serializable.

Isolation level can be specified at the bean or method level. Therefore, it is possible to configure different isolation settings for various methods. This is an advantage when some methods require higher isolation than others, and can be used to achieve maximum performance while maintaining integrity requirements. However, isolation cannot change between method calls within a single enterprise bean transaction. A runtime exception will be thrown in this case.

Isolation levels: Serializable
This level prohibits the following types of reads:

- **Dirty reads:** A transaction reads a database row containing uncommitted changes from a second transaction.
- **Nonrepeatable reads:** One transaction reads a row, a second transaction changes the same row, and the first transaction rereads the row and gets a different value.
- **Phantom reads:** One transaction reads all rows that satisfy an SQL WHERE condition, a second transaction inserts a row that also satisfies the WHERE condition, and the first transaction applies the same WHERE condition and gets the row inserted by the second transaction.

Repeatable Read
This level prohibits dirty reads and nonrepeatable reads, but it allows phantom reads.

Read Committed
This level prohibits dirty reads, but allows nonrepeatable reads and phantom reads.

Read Uncommitted
This level allows dirty reads, nonrepeatable reads, and phantom reads.

The container uses the transaction isolation level attribute as follows:

- Session beans and entity beans with bean-managed persistence (BMP).
  For each database connection used by the bean, the container sets the transaction isolation level at the start of each transaction unless the bean explicitly sets the isolation level on the connection.
- Entity beans with container-managed persistence (CMP).
  The container generates database access code that implements the specified isolation level.

Method extensions - Access intent: WebSphere Application Server enterprise bean method extensions provide settings to specify individual enterprise bean methods as read-only. This setting denotes whether the method can update entity attribute data (or invoke other methods that can update data in the same transaction).

Guide: By default, all enterprise bean methods are assumed to be "update" methods. This results in EJB Entity data always being persisted back to the database at the close of the enterprise bean transaction. Marking enterprise methods as Access Intent Read that do not update entity attributes, provides a significant performance improvement by allowing the WebSphere Application Server EJB container to skip the unnecessary database update.

A behavior for "finder" methods for CMP Entity EJBs is available. By default, WebSphere Application Server will invoke a "Select for Update" query for CMP enterprise bean finder methods such as findByPrimaryKey. This exclusively locks the database row for the duration of the enterprise bean transaction. However, if the enterprise bean finder method has been marked as Access Intent Read, the container will not issue the "For Update" on the select resulting in only a read lock on the database row.

Container transactions: The container transaction properties specifies how the container manages transaction scopes when delegating invocation to the enterprise bean individual business method. The legal values are:

- Never
- Mandatory
- Requires New
- Required
- Supports
- Not Supported
- Bean Managed

Guide: Container transaction attribute can be specified individually for one or more enterprise bean methods. Enterprise bean methods not requiring transactional behavior can be configured as **Supports** to reduce container transaction management overhead.

Legal values: Never
This legal value directs the container to invoke bean methods without a transaction context. If the client invokes a bean method from within a transaction context, the container throws the `java.rmi.RemoteException` exception.

If the client invokes a bean method from outside a transaction context, the container behaves in the same way as if the **Not Supported** transaction attribute was set. The client must call the method without a transaction context.

Mandatory
This legal value directs the container to always invoke the bean method within the transaction context associated with the client. If the client attempts to invoke the bean method without a transaction context, the container throws the `javax.jts.TransactionRequiredException` exception to the client. The transaction context is passed to any enterprise bean object or resource accessed by an enterprise bean method.

Enterprise bean clients that access these entity beans must do so within an existing transaction. For other enterprise beans, the enterprise bean or bean method must implement the **Bean Managed** value or use the **Required** or **Requires New** value. For non-enterprise bean EJB clients, the client must invoke a transaction by using the javax.transaction.UserTransaction interface.

Requires New
This legal value directs the container to always invoke the bean method within a new transaction context, regardless of whether the client invokes the method within or outside a transaction context. The transaction context is passed to any enterprise bean objects or resources that are used by this bean method.

Required
This legal value directs the container to invoke the bean method within a transaction context. If a client invokes a bean method from within a transaction context, the container invokes the bean method within the client transaction context. If a client invokes a bean method outside a transaction context, the container creates a new transaction context and invokes the bean method from within that context. The transaction context is passed to any enterprise bean objects or resources that are used by this bean method.

Supports
This legal value directs the container to invoke the bean method within a transaction context if the client invokes the bean method within a transaction. If the client invokes the bean method without a transaction context, the container invokes the bean method without a transaction context. The transaction context is passed to any enterprise bean objects or resources that are used by this bean method.

Not Supported
This legal value directs the container to invoke bean methods without a transaction context. If a client invokes a bean method from within a transaction context, the container suspends the association between the transaction and the current thread before invoking the method on the enterprise bean instance. The container then

resumes the suspended association when the method invocation returns. The suspended transaction context is not passed to any enterprise bean objects or resources that are used by this bean method.

Bean Managed
This value notifies the container that the bean class directly handles transaction demarcation. This property can be specified only for session beans, not for individual bean methods.

Web module

Web application - Distributable: The distributable flag for J2EE Web applications specifies that the Web application is programmed to be deployed in a distributed servlet container.

Guide: Web applications should be marked as distributable if, and only if, they will be deployed in a WebSphere Application Server cluster or cloned environment.

Web application - Reload interval: Reload interval specifies a time interval, in seconds, in which the Web application file system is scanned for updated files, such as servlet class files or JSPs.

Guide: Reload interval can be defined at different levels for various application components. Generally, the reload interval specifies the time the application server will wait between checks to see if dependent files have been updated and need to be reloaded. Checking file system time stamps is an expensive operation and should be reduced. The default of 3 seconds is good for a test environment, because the Web site can be updated without restarting the application server. In production environments, checking a few times a day is a more common setting.

Web application - Reloading enabled: This specifies whether file reloading is enabled.

Web application - Web components - Load on startup: Indicates whether a servlet is to be loaded at the startup of the Web application. The default is false.

Guide: Many servlets perform resource allocation and other up-front processing in the servlet init() method. These initialization routines can be costly at runtime. By specifying **Load on startup** for these servlets, processing takes place when the application server is started. This avoids runtime delays, which can be encountered on a servlets initial access.

Tuning Java memory

The following section focuses on tuning Java memory. Enterprise applications written in Java involves complex object relationships and utilize large numbers of objects. Although Java automatically manages memory associated with an object's life cycle, understanding the application's usage patterns for objects is important. In particular, ensure the following:

- The application is not over-utilizing objects
- The application is not leaking objects (that is, memory)
- The Java heap parameters are set to handle the use of objects

Understanding the effect of garbage collection is necessary to apply these management techniques.

The garbage collection bottleneck

Examining Java garbage collection can give insight into how the application is utilizing memory. Garbage collection is a Java strength. By taking the burden of memory management away from the application writer, Java applications are more robust than applications written in languages that do not provide garbage collection. This robustness applies as long as the application is not abusing objects. Garbage collection normally consumes anywhere from 5 to 20% of the total execution time of a properly functioning application. If not managed, garbage collection can be one of the biggest bottlenecks for an application, especially when running on SMP server machines.

The garbage collection gauge

You can use garbage collection to evaluate application performance health. By monitoring garbage collection during the execution of a fixed workload, users gain insight as to whether the application is over-utilizing objects. Garbage collection can even be used to detect the presence of memory leaks.

Use the garbage collection and heap statistics in Tivoli Performance Viewer to evaluate application performance health. By monitoring garbage collection, memory leaks and overly-used objects can be detected.

For this type of investigation, set the minimum and maximum heap sizes to the same value. Choose a representative, repetitive workload that matches production usage as closely as possible, user errors included. Allowing the application to run several minutes until the application state stabilizes is important.

To ensure meaningful statistics, run the fixed workload until the state of the application is steady. Reaching this state usually takes several minutes.

Detecting over-utilization of objects

To see if the application is overusing objects, look in Tivoli Performance Viewer at the counters for the JVMPI profiler. The average time between garbage collection calls should be 5 to 6 times the average duration of a single garbage collection. If not, the application is spending more than 15% of its time in garbage collection. Also, look at the numbers of freed, allocated and moved objects.

If the information indicates a garbage collection bottleneck, there are two ways to clear the bottleneck. The most cost-effective way is to optimize the application is to implement object caches and pools. Use a Java profiler to determine which objects to target. If the application cannot be optimized, adding memory, processors and clones might help. Additional memory allows each clone to maintain a reasonable heap size. Additional processors allow the clones to run in parallel.

Detecting memory leaks

Memory leaks in Java are a dangerous contributor to garbage collection bottlenecks. Memory leaks are more damaging than memory overuse, because a memory leak ultimately leads to system instability. Over time, garbage collection occurs more frequently until finally the heap is exhausted and Java fails with a fatal `Out of Memory` exception. Memory leaks occur when an unneeded object has references that are never deleted. This most commonly occurs in collection classes, such as Hashtable, because the table itself always has a reference to the object, even after real references have been deleted.

High workload often causes many applications to crash immediately after being deployed in the production environment. This situation is especially true for leaking applications where the high workload accelerates the magnification of the leakage and a memory allocation failure occurs.

Memory leak testing relates to magnifying numbers. Memory leaks are measured in terms of the amount of bytes or kilobytes that cannot be garbage collected. The delicate task is to differentiate these amounts from the expected sizes of useful and unusable memory. This task is achieved more easily if the numbers are magnified, resulting in larger gaps and easy identification of inconsistencies. The following is a list of important conclusions about memory leaks:

- **Long-running test**

  Memory leak problems can manifest only after a period of time, therefore, memory leaks are found when during long-running tests. Short runs can lead to false alarms. One of the problems in Java is whether to say that a memory leak is occurring when memory usage has seemingly increased either abruptly or monotonically in a given period. These kind of increases can be valid, and the objects created can be referenced at a much later time. In other words, how do you differentiate the delayed use of objects from completely unused objects? By running applications long enough, you will get a higher confidence for whether the delayed use of objects is actually occurring. Because of this, memory leak testing cannot be integrated with some other types of tests, such as functional testing, that occur earlier in the process. However, tests such as stress or durability tests can be integrated.

- **System test**

  Some memory leak problems occur only when different components of a big project are combined and executed. Interfaces between components can produce known or unknown side-effects. System test is a good opportunity to make these conditions happen.

- **Repetitive test**

  In many cases, memory leak problems occur by successive repetitions of the same test case. The goal of memory leak testing is to establish a big gap between unusable memory and used memory in terms of their relative sizes. By repeating the same scenario over and over again, the gap is multiplied in a very progressive way. This testing helps if the amount of leaks caused by an execution of a test case is so minimal that it could hardly be noticed in one run.

  Repetitive tests can be used at the system level or module level. The advantage with modular testing is better control. When a module is designed to keep the private module without creating external side effects such as memory usage, testing for memory leaks can be much easier. First, the memory usage before running the module is recorded. Then, a fixed set of test cases are run repeatedly. At the end of the test run, the current memory usage is recorded and checked for significant changes. Remember, garbage collection must be forced when recording the actual memory usage by inserting System.gc() in the module where you want garbage collection to occur or using a profiling tool forcing the event to occur.

- **Concurrency test**

  Some memory leak problems can occur only when there are several threads running in the application. Unfortunately, synchronization points are very susceptible to producing memory leaks because of the added complication in the program logic. Careless programming can lead to references being kept or unreleased. The incident of memory leaks is often facilitated or accelerated by increased concurrency in the system. The most common way to increase concurrency is to increase the number of clients in the test driver.

Consider the following when choosing which test cases to use for memory leak testing:

- A good test case exercises areas of the application where objects are created. Most of the time, knowledge of the application is required. A description of the

scenario can suggest creation of data spaces, such as adding a new record, creating an HTTP session, performing a transaction and searching a record.

- Look at areas where collections of objects are being used. Typically, memory leaks are composed of objects of the same class. Also, collection classes such as Vector and Hashtable are common places where references to objects are implicitly stored by calling corresponding insertion methods. For example, the get method of a Hashtable object does not remove its reference to the object being retrieved.

Tivoli Performance Viewer helps to find memory leaks. For best results, repeat experiments with increasing duration, like 1000, 2000, and 4000-page requests. The Tivoli Performance Viewer graph of used memory should have a sawtooth shape. Each drop on the graph corresponds to a garbage collection. There is a memory leak if one of the following occurs:

- The amount of memory used immediately after each garbage collection increases significantly. The sawtooth pattern will look more like a staircase.
- The sawtooth pattern has an irregular shape

Also, look at the difference between the number of objects allocated and the number of objects freed. If the gap between the two increases over time, there is a memory leak.

If heap consumption indicates a possible leak during a heavy workload (the application server is consistently near 100% CPU utilization), yet the heap appears to recover during a subsequent lighter or near-idle workload, this is an indication of heap fragmentation. Heap fragmentation can occur when the JVM is able to free sufficient objects to satisfy memory allocation requests during garbage collection cycles, but the JVM does not have the time to compact small free memory areas in the heap into larger contiguous spaces.

Another form of heap fragmentation occurs when small objects (less than 512 bytes) are freed. The objects are freed, but the storage is not recovered, resulting in memory fragmentation.

Heap fragmentation can be avoided by turning on the -Xcompactgc flag in the JVM advanced settings command line arguments. The -Xcompactgc ensures that each garbage collection cycle eliminates fragmentation, but this setting has a small performance penalty.
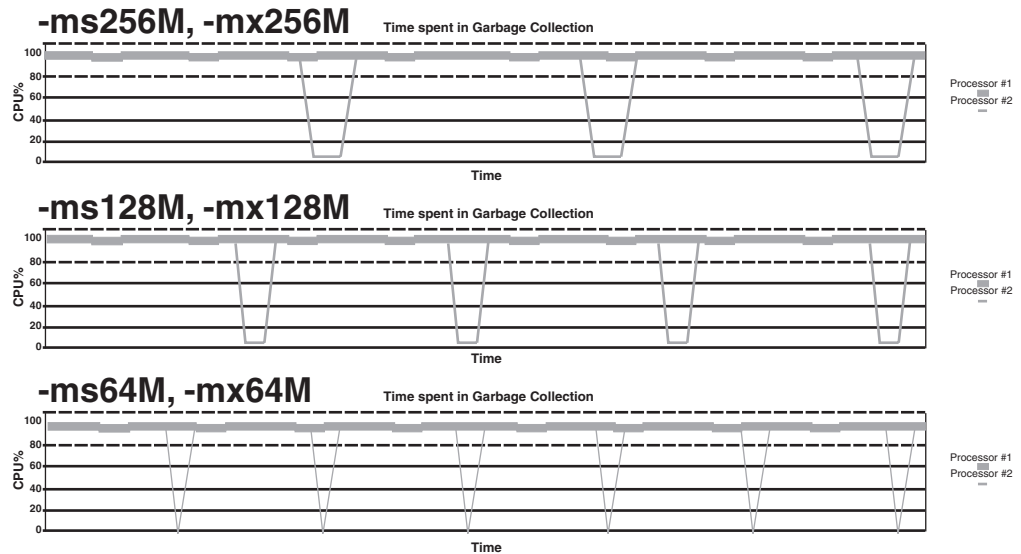
Java heap parameters
The Java heap parameters also influence the behavior of garbage collection. Increasing the heap size allows more objects to be created. Because a large heap takes longer to fill, the application runs longer before a garbage collection occurs. However, a larger heap also takes longer to compact and causes garbage collection to take longer.

For performance analysis, the initial and maximum heap sizes should be equal:
When tuning a production system where the working set size of the Java application is not understood, a good starting value is to let the initial heap size be 25% of the maximum heap size. The JVM will then try to adapt the size of the heap to the working set size of the application.

Varying Java Heap Settings

-ms256M, -mx256M   Time spent in Garbage Collection

CPU%

Processor #1
Processor #2

Time

-ms128M, -mx128M   Time spent in Garbage Collection

CPU%

Processor #1
Processor #2

Time

-ms64M, -mx64M   Time spent in Garbage Collection

CPU%

Processor #1
Processor #2

Time

The illustration represents three CPU profiles, each running a fixed workload with varying Java heap settings. In the middle profile, the initial and maximum heap size are set to 128MB. Four garbage collections occur. The total time in garbage collection is about 15% of the total run. When the heap parameters are doubled to 256MB, as in the top profile, the length of the work time increases between garbage collections. Only three garbage collections, but the length of each garbage collection is also increased. In the third profile, the heap size is reduced to 64MB and exhibits the opposite affect. With a smaller heap, both the time between garbage collections and time for each garbage collection are shorter. For all three configurations, the total time in garbage collection is approximately 15%. This example illustrates an important concept about the Java heap and its relationship to object utilization. There is always a cost for garbage collection in Java applications.

Run a series of test experiments that vary the Java heap settings. For example, run experiments with 128MB, 192MB, 256MB, and 320MB. During each experiment, monitor the total memory usage. If you expand the heap too aggressively, paging can occur. (Use the **vmstat** command or the Windows NT or 2000 Performance Monitor to check for paging.) If paging occurs, reduce the size of the heap or add more memory to the system. When all the runs are finished, compare the following statistics :

* Number of garbage collection calls
* Average duration of a single garbage collection call
* Ratio between the length of a single garbage collection call and the average time between calls

If the application is not over-utilizing objects and has no memory leaks, state of steady memory utilization is reached. Garbage collection also occurs less frequently and for short durations.

If the heap free time settles at 85% or more, consider decreasing the maximum heap size values because the application server and the application are under-utilizing the memory allocated for heap.

Solaris TCP parameters

- **Short description**: Tuning these parameters has a significant performance impact for Solaris. StartupServer.sh sets these:

  ```
  Solaris tcp_time_wait_interval
  Solaris tcp_fin_wait_2_flush_interval
  Solaris tcp_keepalive_interval
  ```

  Many other TCP parameters exist and can affect performance in your environment. For more information about tuning the TCP/IP Stack, see the Web site (Tuning your TCP/IP Stack and More).
- **When to try these parameters**: Try these parameters when you are using WebSphere Application Server on Solaris.

  Before the three TCP parameters were changed, the server stalled during certain peak periods. The **netstat** command showed that many sockets open to port 80 were in the state CLOSE_WAIT or FIN_WAIT_2.

Workload management topology

- **Short description**: WebSphere Application Server provides various Workload Management (WLM) topologies. The following two topologies (name Topology A and B) are examples of workload being sent from one machine:
  - **Topology A** contains a Web server and a WebSphere Application Server plug-in to a cluster of WebSphere Application Servers. Each cluster member contains a Web container and EJB container.
  - **Topology B** includes a Web server, a plug-in, and one Web container to a cluster of EJB containers.

  In both topologies, the Object Request Broker pass-by-reference is selected and the backend database is on a dedicated machine.
- **When to try adjusting:** Topology A has an advantage because the Web container and EJB container are running in a single JVM. In Topology B, the Object Request Broker pass-by-reference option is ignored between the Web container cluster member and the EJB container member. In Topology A, the EJB container uses the same thread passed from the Web container. In other words, the request does not have to be passed from one thread in one JVM to another thread in another JVM.

  Also, if the processor utilization of the cluster member machines is near 100% CPU utilization, additional members could be added. If the Web server box is not running at capacity and the Web container processing is not heavy, try freeing the processor on the other members by moving to Topology B.
- **Actual benefit:** Throughput for Topology A can improve performance from 10-20% more than Topology B. This performance increase can be seen using the J2EE benchmark Trade, which is included with this release.
- **Recommended value:** In the test environment, Topology A had the advantage, however, many factors related to the application and environment can influence results.

Number of connections to DB2

- **Recommended value**: When configuring the data source settings for the databases, ensure the DB2 MaxAppls setting is greater than the maximum

number of connections for the data source. If you are planning to establish clones, the MaxAppls setting needs to be the maximum number of connections multiplied by the number of clones.

The same relationship applies to the session manager number of connections. The MaxAppls setting must be at least as high as the number of connections. If you are using the same database for session and data sources, MaxAppls needs to be the sum of the number of connection settings for the session manager and the data sources.

MaxAppls = (# of connections set for data source + # of connections in session manager) x # of clones

After calculating the MaxAppls settings for the WAS database and each of the application databases, ensure that the MaxAgents setting for DB2 is equal to or greater than the sum of all of the MaxAppls.

```
MaxAgents = sum of MaxAppls for all databases
```

- **Related parameters**: See DB2 MaxAppls and DB2 MaxAgents

Individual performance parameters

As mentioned previously, tuning various system components strongly affects on the performance of WebSphere Application Server. This section discusses how to set the parameters of individual components in order to bring the system to an optimum level of usage.

Hardware

This section discusses considerations for selecting and configuring the hardware on which the application servers will run.

Processor speed

- **Short description**: Ideally, other bottlenecks have been removed where the processor is waiting on events like input/output and application concurrency. In this case, increasing the processor speed often helps throughput and response times.

System memory

- **Short description**: Increasing memory to prevent the system from paging memory to disk is likely to improve performance.

  Allow at least 256MB memory for each processor.
- **When to try adjusting**: Try adjusting the parameter when the system is paging (and processor utilization is low because of the paging).
- **Recommended value:** The minimum value is 512MB.

Networks

- **Short description**: Run network cards and network switches at full duplex. Running at half duplex decreases performance.
  Verify the network speed can accommodate the required throughput. Make sure that 100MB is in use on 10/100 Ethernet networks.

See the white paper (WebSphere Application Server Admin Best Practices for Performance and Scalability) for more information regarding hostname resolution on the administrative client host.

Operating system settings

This section discusses considerations for tuning the operating systems in the server environment.

Windows NT or 2000 TCP/IP parameters

Windows NT or 2000 TcpTimedWaitDelay:
- **Short description:** Determines the time that must elapse before TCP can release a closed connection and reuse its resources. This interval between closure and release is known as the TIME_WAIT state or 2MSL (twice the maximum segment lifetime) state. During this time, reopening the connection to the client and server cost less than establishing a new connection. Reducing the value of this entry allows TCP to release closed connections faster, providing more resources for new connections.
- **When to try adjusting:** If the application running requires rapid release and creation of new connections, and there is a low throughput due to many connections sitting in TIME_WAIT.
- **How to view or set:**
  1. Using the **regedit** command, access HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\ Services\TCPIP\Parameters and create a new REG_DWORD named TcpTimedWaitDelay.
  2. Set the value to decimal 30, which is Hex 0x0000001e.
  3. Restart the system.
  4. By using the **netstat** command, you will be able to see that there are fewer connections in TIME_WAIT.
- **Default value:** 0xF0 (240 seconds = 4 minutes)
- **Recommended value:** The minimum value of 0x1E (30 seconds).
- **Related parameters:** Windows NT or 2000 MaxUserPort

Windows NT or 2000 MaxUserPort:
- **Short description:** Determines the highest port number TCP can assign when an application requests an available user port from the system.
- **How to view or set:**
  1. Using the **regedit** command, access HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\ Services\TCPIP\Parameters and create a new REG_DWORD named MaxUserPort.
  2. Restart the system.
- **Recommended value:** At least decimal 32768.
- **Related parameters:** Windows NT or 2000 TCP/IP parameters

Note: These two parameters should be used together when tuning WebSphere Application Server on a Windows NT or 2000 operating system.

AIX (4.3.3 and 5.1)

AIX file descriptors (ulimit):
- **Short description**: Specifies the number of open files permitted.
- **When to try adjusting**: The default setting is typically sufficient for most applications. If the value set for this parameter is too low, a Memory allocation

error is displayed. This error occurs when a cluster of four clones are run on a S80 24-way and the **ulimit** value needs to be changed to unlimited.

- **How to view or set**: Check the UNIX reference pages on **ulimit** for the syntax for different shells. For the KornShell shell (ksh), to set ulimit to 2000, issue the following command:

  ```
  ulimit -n 2000
  ```

  For large SMP machines with clones, issue the following command:

  ```
  ulimit -unlimited
  ```

  Use the command **ulimit -a** to display the current values for all limitations on system resources.
- **Default value**: For AIX, the default setting is 2000.

Solaris Operating Environment

Solaris file descriptors (ulimit):
- **Short description**: Specifies the number of open files permitted.
- **When to try adjusting**: If the value of this parameter is too low, a `Too many files open` error displays in the WebSphere Application Server stderr.log.
- **How to view or set**: Check the UNIX reference pages on **ulimit** for the syntax for different shells. For KornShell (ksh) the command is:

  ```
  ulimit -n 1024
  ```

  Use **ulimit -a** to display the current values for all limitations on system resources.
- **Default value**: The WebSphere Application Server startupServer.sh script sets this parameter to 1024 if its value is less than 1024.

Solaris TCP_TIME_WAIT_INTERVAL:
- **Short description**: This parameter tells TCP how long to keep closed connection control blocks. After the applications complete the TCP connection, the control blocks are kept for the specified time.
- **When to try adjusting**: When high connection rates occur, a large backlog of the TCP connections build up and can slow server performance.

  The server can stall during certain peak periods. If this occurs, The **netstat** command will show that many of the sockets opened to port 80 were in the CLOSE_WAIT or FIN_WAIT_2 state. Visible delays have occurred for up to four minutes, during which the server did not send any responses, but CPU utilization stayed high, with all of the activity in system processes.
- **How to view or set**: Use the **get** command to determine the current interval and the **set** command to specify an interval of 60 seconds. For example:

  ```
  ndd -get /dev/tcp tcp_time_wait_interval
  ndd -set /dev/tcp tcp_time_wait_interval 60000
  ```
- **Default value**: The Solaris default time wait interval is 2400000 milliseconds.
- **Recommended value**: The TCP_TIME_WAIT_INTERVAL parameter can be set as low as 30000 milliseconds. As a starting point, the WebSphere Application Server startupServer.sh script sets it to 60000ms.
- **Related parameters**: See Solaris TCP parameters

Solaris TCP_FIN_WAIT_2_FLUSH_INTERVAL:

- **Short description**: The timer interval prohibiting a connection in FIN_WAIT_2 to remain in that state.
- **When to try adjusting**: When high connection rates occur, a large backlog of TCP connections accumulate and can slow server performance.

  The server can stall during peak periods. Using the **netstat** command indicated that many of the sockets opened to port 80 were in CLOSE_WAIT or FIN_WAIT_2 state. Visible delays have occurred for as many as four minutes, during which the server did not send any responses, but CPU utilization stayed high, with all of the activity in system processes.
- **How to view and set**: Use the following commands to determine the current interval or to set the interval to 67.5 seconds:

  ```
  ndd -get /dev/tcp tcp_fin_wait_2_flush_interval
  ndd -set /dev/tcp tcp_fin_wait_2_flush_interval 67500
  ```
- **Default value**: The Solaris default is 675000
- **Recommended value**: 67500
- **Related parameters**: See Solaris TCP parameters

Solaris TCP_KEEPALIVE_INTERVAL:
- **Short description**: The timer interval prohibiting an active connection from staying in ESTABLISHED state if one of the peers never responds.
- **When to try adjusting:** If you are concerned with failed communications from clients or peers, this value determines how long a connection will stay open.
- **How to view or set**: Use the following commands to determine the current value or to set the value to 300 seconds:

  ```
  ndd -get /dev/tcp tcp_keepalive_interval
  ndd -set /dev/tcp tcp_keepalive_interval 300000
  ```
- **Default value**: 7200000
- **Recommended Value**: 300000
- **Related parameters**: See Solaris TCP parameters

Other Solaris TCP parameters: Customers have reported success with modifying other Solaris TCP parameters, including the following:

```
tcp_conn_req_max_q
tcp_comm_hash_size
tcp_xmit_hiwat
```

Although significant performance differences have not been seen after raising these settings, the system might benefit.
- **Related parameters**: See Solaris TCP parameters

Solaris kernel semsys:seminfo_semume:
- **Short description**: The semsys:seminfo_semume kernel tuning parameter limits the Max Semaphore undo entries per process and needs to be greater than the default (10 on Solaris 7). Because this setting specifies a maximum value, the parameter does not cause any additional memory to be used unless it is needed.
- **How to view or set**: This value is displayed as SEMUME if the **/usr/sbin/sysdef** command is run. There can possibly be an entry in the /etc/system file for this tuning parameter.

  Set via the /etc/system entry:

  ```
  set semsys:seminfo_semume = 1024
  ```
- **Default value**: 10 on Solaris 7

Solaris kernel semsys:seminfo_semopm:

- **How to view or set**: This setting is displayed SEMOPM if the **/usr/sbin/sysdef** command is run. There can possibly be an entry in the /etc/system file for this tuning parameter.

  Set via the /etc/system entry:

      semsys:seminfo_semopm = 200

Setting the virtual page size for WebSphere Application Server JVM to 64KB:

- **How to view or set**: The command is entered as follows:

      chatr +pi64M +pd64M /opt/WebSphere/AppServer/java/bin/PA_RISC2.0/native_threads/java

  The command output provides the current operating system characteristics of the process executable.

- **When to try adjusting:**
- **Default value:**

The Web server

WebSphere Application Server provides plug-ins for several Web server brands and versions. Each Web server operating system combination has specific tuning parameters that affect the application performance.

This section discusses the performance tuning settings associated with the Web servers.

Web server configuration reload interval

- **Short description**: WebSphere Application Server administration tracks a variety of configuration information about WebSphere Application Server resources. Some of this information, such as URIs pointing to WebSphere Application Server resources, needs to be understood by the Web server. This configuration data is pushed to the Web server through the WebSphere Application Server plug-in at intervals specified by this parameter. Periodic updates allow new servlet definitions to be added without having to restart any of the WebSphere Application Server servers. However, the dynamic regeneration of this configuration information is costly in terms of performance.
- **When to try adjusting**: In a stable production environment.
- **How to view or set**: This parameter, <RefreshInterval=*xxxx*>, where *xxxx* is the number of seconds, is specified in the websphere_root/config/plug-in.xml file.
- **Default value:** The default reload interval is 60 seconds.

IBM HTTP Server (IHS) - AIX and Solaris

The IBM HTTP Server (IHS) is a multi-process, single-threaded server. For more information about tuning IHS, see the Web page (Hints on Running a high-performance Web server).

Sun ONE Web server, Enterprise Edition (formerly iPlanet) - Solaris

The default configuration of the Sun ONE Web server, Enterprise Edition provides a single-process, multi-threaded server.

Active threads

- **Short description:** After the server reaches the limit set with this parameter, the server stops servicing new connections until it finishes old connections.

- **When to try adjusting:** If this setting is too low, the server can become throttled, resulting in degraded response times.

  To tell if the Web server is being throttled, consult its perfdump statistics. Look at the following data:

  - **WaitingThreads count:** If WaitingThreads count is getting close to zero, or is zero, the server is not accepting new connections.
  - **BusyThreads count:** If the WaitingThreads count is close to zero, or is zero, BusyThreads is probably very close to its limit.
  - **ActiveThreads count:** If ActiveThreads count close to its limit, the server is probably limiting itself.

- **How to view or set:** Use the Maximum number of simultaneous requests parameter in the Enterprise Server Manager interface to control the number of active threads within Sun ONE Web server, Enterprise Edition. This setting corresponds to the RqThrottle parameter in the magnus.conf file.

- **Default value:** 512

Microsoft Internet Information Server (IIS) - Windows NT or 2000

IIS permission properties

- **Short description:** The Web server has several properties that dramatically affect the performance of the application server. The default settings are usually acceptable. However, because other products can change the default settings without user knowledge, make sure to check the IIS settings for the Home Directory permissions of the Web server. The permissions should be set to Script and not to Execute. If the permissions are set to Execute, no error messages are returned, but the performance of WebSphere Application Server is decreased.

- **How to view or set:** To check or change these permissions perform the following procedure in the Microsoft management console:
  Select the Web site (usually default Web site). Right-click and select the **Properties** option. Click the **Home Directory** tab.

  To set the permissions of the Home Directory:
  In the Application settings, ensure that the **Script** checkbox is selected in the **Permissions** list and that the **Execute** checkbox is cleared.
  Note: It might be necessary to check the permissions of the sePlugin:
  Expand the Web server. Right-click the **sePlugin** and select **Properties**. Confirm that the **Execute** permissions are set to **Execute**.

Number of expected hits per day

- **Short description:** This parameter controls the memory that IIS allocates for connections.

- **How to view or set:** Using the performance window, set the parameter to More than 100000 in the Web site properties panel of the Microsoft management console.

- **Default value:** Fewer than 100000

ListenBackLog parameter

- **Short description:** If using IIS on Windows NT/2000, you are likely to encounter failed connections under heavy load conditions (typically more than 100+ clients). This condition commonly results from IIS rejecting connections. Alleviate the condition by using the ListenBackLog parameter to increase the number of requests IIS keeps in its queue.

- **When to try adjusting:** If you intermittently experience being unable to locate server error in a Netscape browser when running a heavy load, consider raising this parameter.

- **How to view or set:**
  Use the operating system registry to set the ListenBackLog parameter. At a DOS command prompt, issue the regedit command to access the registry. In the registry window, locate the parameter in the following directory: HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\InetInfo\Parameters\ ListenBackLog Right-click the parameter to modify it. Adjust the setting according to the server load.
- **Default value:** 25 (decimal)
- **Recommended value:** The ListenBackLog parameter can be set as high as 200 without negative impact on performance and an improvement in load handling.

MaxPoolThreads, PoolThreadLimit:

- **Short description:** MaxPoolThreads controls the number of threads per CPU in the thread pool available for IIS to run Common Gateway Interface (CGI) processes (each process takes one thread). PoolThreadLimit specifies the upper limit for MaxPoolThreads. The default thread limit that IIS can create on a machine is twice the number of MB in RAM on a machine (for example, a server with 512MB of RAM is limited to 1024 threads).
- **How to view or set:** MaxPoolThreads and PoolThreadLimit are set using the following registry entries:

  ```
  HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\InetInfo\Parameters\MaxPoolThreads
  HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\InetInfo\Parameters\PoolTheadLimit
  ```

MaxClients:

- **Short description**: The value of the MaxClients parameter can significantly impact the application, particularly if too high. More is not always better. The optimum value depends on the application.
- **How to view or set**: Edit the MaxClients directive in the IBM HTTP Server file httpd.conf, located in the directory *IBM HTTP Server_root_directory/conf.*
- **Default value**: 150
- **Related parameters**: See Adjusting WebSphere Application Server system queues

MinSpareServers, MaxSpareServers, and StartServers:

- **Short description**: These settings affect the performance of the application. For optimum performance, specify the same value for the MaxSpareServers and the StartServers parameters. Specifying similar values reduces the CPU usage for creating and destroying httpd processes. It also pre-allocates and maintains the specified number of processes so that few processes are created and destroyed as the load approaches the specified number of processes (based on MinSpareServers).
- **When to try adjusting:**
- **How to view or set**: Edit the following directives in the file httpd.conf, located in the directory *IBM_HTTP_Server_root_directory/conf*:
  - MinSpareServers
  - MaxSpareServers
  - StartServers
- **Default value**:
  - MinSpareServers 5
  - MaxSpareServers 10
  - StartServers 5

IBM HTTP Server - Linux

MaxRequestsPerChild:

- **Short description**: The MaxRequestsPerChild directive sets the limit on the number of requests that an individual child server process handles. After the number of requests reaches the value set for the MaxRequestsPerChild parameter, the child process dies. If there are no known memory leaks with Apache and Apache's libraries, set this value to zero (0).
- **When to try adjusting**:
- **How to view or set**:
  1. Edit the IBM HTTP server file `httpd.conf` located in the directory *IBM_HTTP_Server_root_directory/conf* (see the MaxRequestsPerChild directive).
  2. Change the value of the parameter.
  3. Save the changes and restart the IBM HTTP server.
- **Default value**: 500

IBM HTTP Server - Windows NT or 2000
The IBM HTTP Server can be easily configured. The default settings are acceptable.

ThreadsPerChild:

- **Short description**: This parameter sets the number of concurrent threads running at any one time within the IBM HTTP Server. Set this value to prevent bottlenecks, allowing just enough traffic through to the application server.
- **How to view or set**:
  1. Edit the IBM HTTP Server file `httpd.conf` located in the directory *IBM_HTTP_Server_root_directory/conf* (see the ThreadsPerChild directive).
  2. Change the value of the parameter.
  3. Save the changes and restart the IBM HTTP server.

  How to view thread utilization: Two ways to find how many threads are being used under load are as follows:
  1. Use the Windows NT or 2000 Performance Monitor:
     To open, click **Start** > **Programs** > **Administrative Tools** > **Performance Monitor**
     In Performance Monitor, click **Edit** > **Add to chart**. Then set the following:
     – Object: IBM HTTP Server
     – Instance: Apache
     – Counter: Waiting for connection
     – To calculate the number of busy threads, subtract the number waiting (Windows NT or 2000 Performance Monitor) from the total available (ThreadsPerChild).
  2. Use IBM HTTP Server server-status (this choice works on all platforms, not just Windows NT or 2000)
     Follow these steps to use IBM HTTP Server server-status:
     a. Edit the IBM HTTP Server file `httpd.conf` as follows:
        – Remove the comment character "#" from the following lines:
          - #LoadModule status_module modules/ApacheModuleStatus.dll
          - #<Location /server-status>
          - #SetHandler server-status
          - #</Location>

b. Save the changes and restart the IBM HTTP server.

c. In a Web browser, go to the following URL and click **Reload** to update status: http://yourhost/server-status. Alternatively, if the browser supports refresh, go to http://yourhost/server-status?refresh=5 to refresh every 5 seconds. You will see 5 requests currently being processed, 45 idle servers.

- **Default value**: 50 (for IBM HTTP Server 1.3.26).
- **Related parameters**: See Adjusting WebSphere Application Server system queues

ListenBackLog:

- **Short description**: When several clients request connections to the IBM HTTP Server, and all threads (see ThreadsPerChild) are being used, a queue exists to hold additional client requests. The ListenBackLog directive sets the length of this pending connections queue. However, if you are using the default Fast Response Cache Accelerator (FRCA) feature, the ListenBackLog directive is not used since FRCA has its own internal queue.
- **How to view or set**: For non-FRCA:
  1. Edit IBM HTTP Server file httpd.conf.
  2. Add or view the ListenBackLog directive.
- **Default value**: For HTTP Server 1.3.26:
  1024 with FRCA enabled
  511 with FRCA disabled (the default)
- **Recommended value**: Use the defaults.

The WebSphere Application Server process

Each WebSphere Application Server process has several parameters influencing application performance. Each application server in WebSphere Application Server is comprised of an EJB container and Web container.

Use the WebSphere Application Server administrative console to configure and tune applications, Web containers, EJB containers, application servers and nodes in the administrative domain.

Adjusting the operating system priority of the WebSphere Application Server process

- **Short description:** Improving the operating system process priority of the application server can help performance. On UNIX systems, a smaller setting has a higher priority. For more information about process priorities, refer to the following publications:
  - For AIX, refer to the AIX Performance Tuning Guide, Versions 3.2 and 4
- **When to try adjusting:** Try adjusting the priority at any time.
- **Actual benefit:** By improving the priority of an application server, noticeable improvements have been seen on AIX. Improvement has also been seen on Solaris.
- **How to view or set:**
  1. In the administrative console, select the application server you are tuning.
  2. Click the **Process Definition** under **Additional Properties**.
  3. Click **Process Execution** under **Additional Properties**.
  4. Specify the value in the **Process Priority** field and click **Apply**.
  5. Save the changes and restart the application server.

How to see parameter utilization: On UNIX, use the command **ps -efl** to see the current process priority.
- **Default value:** 20
- **Recommended value:** 0

Web containers

To route servlet requests from the Web server to the Web containers, the product establishes a transport queue between the Web server plug-in and each Web container.

Thread pool:

- **Short description**: Use the maximum thread size parameter to specify the maximum number of threads that can be pooled to handle requests sent to the Web container. Requests are sent to the Web container through any of the HTTP transports.

- **How to view or set**:
  1. In the administrative console, select the application server you are tuning.
  2. Click **Servers** > *server* > **Web Container** > **Thread Pool**.
  3. Enter the desired maximum number of maximum threads in the **Maximum Size** field.
  4. Select or deselect the **Growable Thread Pool** checkbox. See Adjusting the queues in WebSphere Application Server for more information.
  5. Click **Apply** or **OK**.
  6. Click **Save**.

  Tivoli Performance Viewer displays a metric called Percent Maxed that determines the amount of time that the configured threads are in use. If this value is consistently in the double-digits, the Web container could be a bottleneck and the number of threads should be increased.

- **Default value**: 50
  Note: For Linux systems, the recommended value is 25.
- **Related parameters**: See Adjusting WebSphere Application Server's system queues and Prepared statement cache size

MaxKeepAliveConnections:

- **Short description:** This parameter describes the maximum number of concurrent connections to the Web container that are allowed to be kept alive, that is, to be processed in multiple requests. The Web server plug-in keeps connections open to the application server as long as it can. However, if the value of this property is too small, performance is negatively impacted because the plug-in has to open a new connection for each request instead of sending multiple requests through one connection. The application server might not accept a new connection under a heavy load if there are too many sockets in TIME_WAIT state.

  If all client requests are going through the Web server plug-in and there are many TIME_WAIT state sockets for port 9080, the application server is closing connections prematurely, which decreases performance. The application server will close the connection from the plug-in, or from any client, for any of the following reasons:
  1. The client request was an HTTP 1.0 request when the Web server plug-in always sends HTTP 1.1 requests.

2. The maximum number of concurrent keep-alives was reached. A keep-alive must be obtained only once for the life of a connection, that is, after the first request is completed, but before the second request can be read.

3. The maximum number of requests for a connection was reached, preventing denial of service attacks in which a client tries to hold on to a keep-alive connection forever.

4. A time out occurred while waiting to read the next request or to read the remainder of the current request.

- **How to view or set:**
  1. Open the administrative console.
  2. Click **Servers** > **Application Servers** > **Web Container** > **HTTP Transports** .
  3. Click the *port_number* link in the Host column.
  4. Click **Custom Properties** > **New**.
  5. Enter the MaxKeepAliveConnections name in the **Name** field.
  6. Enter the value in the **Value** field
  7. Click **Apply** or **OK**.
  8. Click **Save**.

- **Recommended value:** The value should be at least 90% of the maximum number of threads in the Web container thread pool. If it is 100% of the maximum number of threads in the Web container thread pool, all the threads could be consumed by keep-alive connections leaving no threads available to process new connections.

MaxKeepAliveRequests:

- **Short description:** The maximum number of requests allowed on a single keep-alive connection. This parameter can help prevent denial of service attacks when a client tries to hold on to a keep-alive connection. The Web server plug-in keeps connections open to the application server as long as it can, providing optimum performance.

- **How to view or set:**
  1. Open the administrative console.
  2. Click **Servers** > **Application Servers** *server* > **Web Container** > **HTTP Transports** .
  3. Click the *port_number* link in the Host column.
  4. Click **Custom Properties** > **New**.
  5. Enter the MaxKeepAliveRequests name in the **Name** field.
  6. Enter the value in the **Value** field
  7. Click **Apply** or **OK**.
  8. Click **Save**.

- **Recommended value:** A good starting value is 100. If the application server requests are received from the plug-in only, increase this parameter's value.

URL invocation cache:

- **Short description**: The invocation cache holds information for mapping request URLs to servlet resources.

  A cache of the requested size is created for each thread. The number of threads is determined by the Web container maximum thread size setting.

Note: A larger cache uses more of the Java heap, so you might need to increase maximum Java heap size. For example, if each cache entry requires 2KB, maximum thread size is set to 25, and the URL invocation cache size is 100; then 5MB of Java heap are required.

- **When to try adjusting**: If more than 50 unique URLs are actively being used (each JSP is a unique URL), increase this parameter.
- **How to view or set**: The size of the cache can be specified for the application server along with other JDK parameters by:
  1. In the administrative console, click the application server you are tuning.
  2. Click **JVM Setting**.
  3. Click **Add** in the **System Properties** section.
  4. Add the name invocationCacheSize and a value of 50.
  5. Click **Apply** to ensure that the changes are saved.
  6. Stop and restart the application server.
- **Default value**: 50
- **Related parameters**: See Heap size settings

Allow thread allocation beyond maximum

- **Short description:** When this option is selected, more Web container threads can be allocated than specified in the maximum thread size field.
- **How to view or set:**
  1. In the administrative console, select the application server you are tuning.
  2. Click **Web Container Service** under **Additional Properties**.
  3. Click **Thread Pool** under **Additional Properties**.
  4. Select the checkbox **Growable thread pool**.
  5. Click **Apply** to ensure the changes are saved.
  6. Stop and restart the application server.
- **Default value:** The default value setting is "unchecked" (the thread pool can not grow beyond the value specified for the maximum thread size).
- **Recommended value:** This option is intended to handle brief loads beyond the configured maximum thread size. However, use caution when selecting this option because too many threads can cause the system to overload.

Dynamic cache service: The dynamic cache service improves performance by caching the output of servlets, commands and Java Server Pages (JSP) files. WebSphere Application Server consolidates several caching activities, including servlets, Web services, and WebSphere commands into one service called the dynamic cache. These caching activities work together to improve application performance, and share many configuration parameters, which are set in an application server's dynamic cache service.

The dynamic cache works within an application server Java Virtual Machine (JVM), intercepting calls to cacheable objects, for example, through a servlet's service() method or a command's execute() method, and either stores the object's output to or serves the object's content from the dynamic cache. Because J2EE applications have high read-write ratios and can tolerate small degrees of latency in the currency of their data, the dynamic cache can create an opportunity for significant gains in server response time, throughput, and scalability.

See the InfoCenter article (Improving performance through the dynamic cache service) for more information.

Security

This section discusses how various settings related to security affect performance.

Disabling security:

- **Short description**: Security is a global setting. When security is enabled, performance can be decreased between 10-20%. Therefore, disable security when not needed.
- **How to view or set**: In the administrative console, click **Security** > **Global Security**. The checkboxes **Enabled** and Enforce **Java 2 Security** control global security settings.
- **Default value**: By default, security is not enabled.

Fine-tune the security cache time out for the environment:

- **Short description**: If WebSphere Application Server security is enabled, the security cache time out can influence performance. The time out parameter specifies how often to refresh the security-related caches.

  Security information pertaining to beans, permissions, and credentials is cached. When the cache time out expires, all cached information becomes invalid. Subsequent requests for the information result in a database lookup. Sometimes, acquiring the information requires invoking a Lightweight Directory Access Protocol(LDAP)-bind or native authentication. Both invocations are relatively costly operations for performance.

  Determine the best trade-off for the application, by looking at usage patterns and security needs for the site.
- **Actual benefit observed**: In a 20-minute performance test, when the cache time out was set so that a time out did not occur, a 40% performance improvement was achieved.
- **How to view or set**: In the administrative console, click **Security** > **Global Security** > **Cache Timeout**.
- **Default value:** The default is 600.

Security cache types and sizes (system parameters):   The following system properties determine the initial size of the primary and secondary Hashtable caches, which affect the frequency of rehashing and the distribution of the hash algorithms. The larger the number of available hash values, the less likely a hash collision will occur, and more likely a slower retrieval time. If several entries compose a Hashtable cache, creating the table in a larger capacity allows the entries to be inserted more efficiently rather than letting automatic rehashing determine the growth of the table. Rehashing causes every entry to be moved each time.

com.ibm.websphere.security.util.LTPAAuthCacheSize

- **Short description:** This cache stores basic authentication credentials at the Security Server. Whenever an Lightweight Third Party Authentication (LTPA) token expires, a new token is generated from the basic authorization credentials in this cache. If no basic authorization credentials exist, the requesting browser must send the basic authorization credentials to the Security Server. The browser will prompt the user for user ID and password if no cookie exists containing the credentials.

com.ibm.websphere.security.util.LTPATokenCacheSize

- **Short description:** This cache stores LTPA credentials in the cache using the LTPA token as a lookup value. When using an LTPA token to log in, the LTPA

credential is created at the Security Server for the first time. This cache prevents the need to go to the Security Server on subsequent logins using an LTPA token.

com.ibm.websphere.security.util.CredentialCacheSize

- **Short description:** Given the user ID and password for login, this cache returns the concrete credential object, either Local OS or LTPA, without the need to repeat authentication at the Security Server. If the credential object has expired, the need to repeat authentication is required.

com.ibm.websphere.security.util.LTPAValidationCacheSize

- **Short description:** Given the credential token for login, this cache returns the concrete LTPA credential object, without the need to revalidate at the Security Server. If the token has expired, the need to revalidate is required.

com.ibm.websphere.security.util.PermissionCacheSize

- **Short description:** This cache holds the WebSphere Application Server permission objects retrieved when a getGrantedPermissions method is called. If access to the same resource by the same principal occurs again, the permissions will be retrieved rapidly from the cache instead of going to the repository on the administrative server. This cache is common to both enterprise bean and Web-granted permissions.

com.ibm.websphere.security.util.AdminBeanCacheSize

- **Short description:** This cache stores information, including the required permissions, about enterprise beans that have been deployed in the administrative server.

com.ibm.websphere.security.util.BeanCacheSize

- **Short description:** This cache stores information, including the required permissions and RunAs mode, about enterprise beans that have been deployed in a container on the application server.

Configure SSL sessions appropriately:

- **Short description**: The SSLV3Timeout value specifies the time interval after which SSL sessions are renegotiated. This is a high setting, and modification probably does not provide any significant impact. By default, it is set to 9600 seconds.

  The Secure Association Service (SAS) feature establishes an SSL connection only if it goes out of the JVM to another JVM. Therefore, if all the beans are co-located within the same JVM, the SSL used by SAS is not expected to hinder performance.
- **How to view or set**: Modify the SSLV3Timeout and other SAS properties by editing `sas.server.props` and `sas.client.props` files. The files are located in the *product_installation_root*\properties directory, where *product_installation_root* is the directory where WebSphere Application Server is installed.
- **Default value:** The default is 9600 seconds.

Object Request Broker (ORB)

Several settings are available for controlling internal Object Request Broker (ORB) processing. You can use these to improve application performance in the case of applications containing enterprise beans.

You can change these settings for the default server or any application server configured in the administrative domain.

To change the settings, click **Servers** > **Application Servers**. Then, click **ORB Service** from **Additional Properties**.

Pass-by-value versus pass-by-reference (NoLocalCopies):
- **Short description**: For EJB 1.1 beans, the EJB 1.1 specification states that method calls are to be pass-by-value. For every remote method call, the parameters are copied onto the stack before the call is made. This can be expensive. The pass-by-reference, which passes the original object reference without making a copy of the object, can be specified.

  For EJB 2.0 beans, interfaces can be local or remote. For local interfaces, method calls are pass-by-reference, by default.
- **Actual benefit observed**: If the EJB client and EJB server are installed in the same WebSphere Application Server instance, and the client and server use remote interfaces, specifying pass-by-reference can improve performance up to 50%.
- **When to try adjusting**: Pass-by-reference helps performance only in the case where non-primitive object types are being passed as parameters. Therefore, int and floats are always copied, regardless of the call model.

  WARNING: Pass-by-reference can be dangerous and can lead to unexpected results. If an object reference is modified by the remote method, the change might be seen by the caller.
- **How to view or set**: Click **Servers** > **Application Servers**. Then, click **ORB Service** from **Additional Properties**.
- Select **Pass by Reference**.
- Click **OK**.
- Click **Apply** to save the changes.
- Stop and restart the application server.
- **Default value**: Pass-by-value for remote interfaces, pass-by-reference for EJB 2.0 local interfaces.

If the application server expects a large workload for enterprise bean requests, the ORB configuration is critical. Take note of the following properties:

com.ibm.CORBA.ServerSocketQueueDepth:
- **Short description:** This property corresponds to the length of the TCP/IP stack listen queue and prevents WebSphere Application Server from rejecting requests when there is not space in the listen queue.
- **When to try adjusting:** If there are many simultaneous clients connecting to the server-side ORB, this parameter can be increased to support the heavy load up to 1000 clients.
- **How to view or set:** To set the property, follow these steps:
  1. Click **Servers** > **Application Servers**.
  2. Click the application server you want to tune.
  3. Click **Process Definition** under **Additional Properties**.
  4. Click **Java Virtual Machine** under **Additional Properties**.
  5. Type

     `-Dcom.ibm.CORBA.ServerSocketQueueDepth=200.`

     in the **Generic JVM Properties** field.
- **Default value:** 50

com.ibm.CORBA.MaxOpenConnections and Object Request Broker connection cache maximum:

- **Short description:** This property has two names and corresponds to the size of the ORB connection table. The property sets the standard for the number of simultaneous ORB connections that can be processed.
- **When to try adjusting:**If there are many simultaneous clients connecting to the server-side ORB, this parameter can be increased to support the heavy load up to 1000 clients.
- **How to view or set:**
  1. Click **Servers** > **Application Servers**.
  2. Click the application server you want to tune.
  3. Click the **ORB Service** under **Additional Properties**.
  4. Update the **Connection cache maximum** field with **OK**.
  5. Update the Connection cache maximum field and click **OK**.
  6. Click **Apply** to save the changes.
  7. Restart the application server.
- **Default value:** 240

ORB thread pool size:
- **Short description**: This property relates to the size of the ORB thread pool. A worker thread is taken from the pool to process requests from a given connection.
- **When to try adjusting:** If there are many simultaneous clients connecting to the server-side ORB, this parameter can be increased to support the heavy load up to 1000 clients.
- **How to view or set**:
  1. In the administrative console, select the application server you are tuning and then click the **Services** tab.
  2. Click **Servers** > **Application Servers**.
  3. Click the application server you want to tune.
  4. Click the **ORB Service** under **Additional Properties**.
  5. Click **Thread Pool** under **Additional Properties**.
  6. Update the **Maximum Size** field and click **OK**.
  7. Click **Apply** to save the changes.
  8. Restart the application server.
- **Related parameters**: See Adjusting WebSphere Application Server's system queues
- **Default value:** 50

Java Virtual Machines (JVMs)

Tuning the JVM

The JVM offers several tuning parameters affecting the performance of WebSphere Application Servers and application performance.

Sun JDK 1.3 HotSpot -server warm-up
- **Short description:** The HotSpot JVM introduces adaptive JVM technology containing algorithms for optimizing byte code execution over time. The JVM runs in two modes, -server and -client. Performance can be significantly

enhanced if running in -server mode and a sufficient amount of time is allowed for a HotSpot JVM to warm-up by performing continuous execution of byte code.

- **When to try adjusting:** In most cases, -server mode should be run. This produces more efficient run time execution over extended periods. The -client option can be used if a faster startup time and smaller memory footprint are preferred, at the cost of lower extended performance.
- **How to view or set:** The -server option is enabled by default. Follow these steps to change the -client or -server mode:
  1. Click **Servers** > **Application Servers**.
  2. Click the application server you want to tune.
  3. Click the **Process Definition** under **Additional Properties**.
  4. Click **Java Virtual Machine** under **Additional Properties**.
  5. Click **Custom Properties** under **Additional Properties**.
  6. Select **New** and enter *HotSpotOption* in the **Name** field. Enter *-client* or *-server* in the **Value** field.
  7. Restart the application server.
- **Recommended value:** -server

Sun JDK 1.3 HotSpot new generation pool size

- **Short description:** Most garbage collection algorithms iterate every object in the heap to determine which objects to free. The HotSpot JVM introduces generational garbage collection which makes use of separate memory pools to contain objects of different ages. These pools can be garbage collected independently from one another. The sizes of these memory pools can be adjusted. Extra work can be avoided by sizing the memory pools so that short-lived objects will never live through more than one garbage collection cycle.
- **When to try adjusting:** If garbage collection has become a bottleneck, try customizing the generation pool settings.
- **How to view or set:**
-
  1. Click **Servers** > **Application Servers**.
  2. Click the application server you want to tune.
  3. Click the **Process Definition** under **Additional Properties**.
  4. Click **Java Virtual Machine** under **Additional Properties**.
  5. Click **Custom Properties** under **Additional Properties**.
  6. Enter the following values in the **Generic JVM Arguments** field: *-XX:NewSize* (lower bound), *-XX:MaxNewSize* (upper bound)
  7. Restart the application server.
- **Recommended value:** Bound the new generation between 25 to 50% the total heap size.

Just In Time (JIT) compiler

- **Short description**: The Just In Time (JIT) compiler can significantly affect performance.
- **When to try adjusting**: In all cases, run with JIT enabled, which is the default. To disable JIT:
  1. Click **Servers** > **Application Servers**.
  2. Click the application server you want to tune.

3. Click the **Process Definition** under **Additional Properties**.

4. Click **Java Virtual Machine** under **Additional Properties**.

5. Select the checkbox **Disable JIT**.

6. Save your changes.

7. Restart the application server.

Heap size settings

- **Short description**: These parameters can set the maximum and initial heap sizes for the JVM.

  In general, increasing the size of the Java heap improves throughput until the heap no longer resides in physical memory. After the heap begins swapping to disk, Java performance drastically suffers. Therefore, the maximum heap size needs to be low enough to contain the heap within physical memory.

  The physical memory usage must be shared between the JVM and the other applications, such as, the database. For assurance, use a smaller heap, for example 64MB, on machines with less memory.

  Try a maximum heap of 128MB on a smaller machine, that is, less than 1GB of physical memory, 256MB for systems with 2GB memory, and 512MB for larger systems. The starting point depends on the application.

  If performance runs are being conducted and highly repeatable results are needed, set the initial and maximum sizes to the same value. This setting eliminates any heap growth during the run. For production systems where the working set size of the Java applications is not well understood, an initial setting of one-fourth the maximum setting is a good starting value. The JVM will then try to adapt the size of the heap to the working set of the Java application.

- **How to view or set**:
  1. Click **Servers** > **Application Servers**.
  2. Click the application server you want to tune.
  3. Click the **Process Definition** under **Additional Properties**.
  4. Click **Java Virtual Machine** under **Additional Properties**.
  5. Enter values in the **General Properties** field for the following fields: *Initial Heap Size* and *Maximum Heap Size*.
  6. Save your changes.
  7. Restart the application server.

Class garbage collection

- **Short description**: In most cases, run with class garbage collection turned on. This is the default.

- **How to view or set**: Disabling class garbage collection enables more class reuse, which, in some cases, has resulted in small performance improvements.

  Use the **Generic JVM Arguments** of the default server or any additional application server you configure in the administrative domain to set the JVM parameters:

  1. Click **Servers** > **Application Servers**.
  2. Click the application server you want to tune.
  3. Click the **Process Definition** under **Additional Properties**.
  4. Click **Java Virtual Machine** under **Additional Properties**.
  5. Enter the value *-Xnoclassgc* in the **Generic JVM Arguments** field.
  6. Save your changes.
  7. Restart the application server.

EJB container

Cache settings

- **Short description:** To determine the cache absolute limit, multiply the number of enterprise beans active in any given transaction by the total number of concurrent transactions expected. Then, add the number of active session bean instances. Use the Tivoli Performance Viewer to view bean performance information.
- **How to view or set**: Edit the EJB container service properties for the application server you are tuning.
- **Default value**: Cache size=2053, Cache clean-up interval=3000

Break CMP enterprise beans into several enterprise bean modules

- **Short description:** The load time for hundreds of beans can be improved by distributing the beans across several JAR files and packaging them to an EAR file. This is faster when the administrative server attempts to start the beans, for example, 8-10 minutes versus more one hour when one JAR file is used.

XML parser selection

- **Short description**: Add XML parser definitions to the `jaxp.properties` file and `xerces.properties` file found in the ${WAS_HOME}/jre/lib directory to help facilitate server startup. The XMLParserConfiguration value might have to be changed as new versions of Xerces are provided.
- **How to view or set**: In both files, insert the following lines (last split for publication):

```
javax.xml.parsers.SAXParserFactory=org.apache.xerces.jaxp.SAXParserFactoryImpl
javax.xml.parsers.DocumentBuildFactory=org.apache.xerces.jaxp.DocumentBuilderFactoryImpl
org.apache.xerces.xni.parser.XMLParserConfiguration=
    org.apache.xerces.parsers.StandardParserConfiguration
```

Data sources

Connection pool size

- **Short description**: When accessing any database, the initial database connection is an expensive operation. WebSphere Application Server supports JDBC 2.0 Standard Extension APIs to provide support for connection pooling and connection reuse. The connection pool is used for direct JDBC calls within the application, as well as for enterprise beans using the database.
- **Actual benefit observed**: When the connection pooling capabilities are used, performance improvements up to 20 times the normal results can be realized.
- **How to view or set**: Use the administrative console to specify the database connection pool options.
- **Default value**: Minimum connection pool size: 1, Maximum connection pool size: 10
- **How to view parameter utilization**: Use the Tivoli Performance Viewer:
  1. Start the Tivoli Performance Viewer.
  2. Select the application server you are tuning.
  3. Expand the database connection pools category.
  4. Select the data source.
  5. Click the data source and set to high. High is necessary to obtain the percentUsed counter.
  6. Click **Run** to start collecting statistics, and observe the following:

- Pool size = Average number of connections in the pool to the database
- Percent used = Average percent of the pool connections in use
- Concurrent waiters = Average number of threads waiting for a connection

  Use Tivoli Performance Viewer to find the optimal number of pool connections that can reduce values for these numbers. If Percent Used is consistently low, consider decreasing the number of connections in the pool.

- **Recommended value**: Better performance is generally achieved if the value for the connection pool size is set lower than the value for the Max Connections in the Web container. Lower settings (10-30 connections) typically perform better than higher (more than 100) settings.

  On UNIX platforms, a separate DB2 process is created for each connection. These processes quickly affects performance on systems with low memory, causing errors.

  Each Entity EJB transaction requires an additional connection to the database specifically to handle the transaction. Be sure to take this into account when calculating the number of data source connections.

  Deadlock can occur if the application requires more than one concurrent connection per thread, **and** the database connection pool is not large enough for the number of threads. Suppose each of the application threads requires two concurrent database connections and the number of threads is equal to the maximum connection pool size. Deadlock can occur when both of the following are true:
  - Each thread has its first database connection, and all are in use.
  - Each thread is waiting for a second database connection, and none would become available since all threads are blocked.

  To prevent the deadlock in this case, the value set for the database connection pool must be at least one higher, one of the waiting threads to complete its second database connection and free up to allow database connections.

  To avoid deadlock, code the application to use, at most, one connection per thread. If the application is coded to require C concurrent database connections per thread, the connection pool must support at least the following number of connections, where T is the maximum number of threads.

      T * (C - 1) + 1

  The connection pool settings are directly related to the number of connections that the database server is configured to support. If the maximum number of connections in the pool is raised, and the corresponding settings in the database are not raised, the application fails and SQL exception errors are displayed in the stderr.log file.

- **Related parameters**: See Adjusting WebSphere Application Server system queues, Prepared statement cache size, and Number of connections to DB2

Prepared statement cache size

- **Short description**: The WebSphere Application Server data source optimizes the processing of prepared statements. A prepared statement is a precompiled SQL statement that is stored in a prepared statement object. This object is used to efficiently execute the given SQL statement multiple times.

  Note: Prepared statements are optimized for handling parametric SQL statements that benefit from precompilation. If the JDBC driver specified in the data source supports precompilation, the creation of the prepared statement will

send the statement to the database for precompilation. Some drivers might not support precompilation and the prepared statement might not be sent until the prepared statement is executed.

- **Actual benefit observed**: In test applications, tuning the prepared statement cache improved throughput by 10-20%.
- **How to view or set**: In the administrative console click **Resources** > **JDBC Providers** > *provider_name* > **Data source** > *data_source_name* > **Connection Pool** > **Statement Cache Size**. The **Statement Cache Size** field should contain a value that is the total cache size, not the size per container.
- **Recommended value**: If the cache is not large enough, useful entries will be discarded to make room for new entries. In general, the more prepared statements your application has, the larger the cache should be. For example, if the application has 5 SQL statements, set the prepared statement cache size to 5, so that each connection has 5 statements.

  Tivoli Performance Viewer can help tune this setting to minimize cache discards. Use a standard workload that represents a typical number of incoming client requests, use a fixed number of iterations, and use a standard set of configuration settings.

  Follow these instructions to use the Tivoli Performance Viewer:
  1. Start Tivoli Performance Viewer.
  2. Click *host* > *server* > **JDBC Connection Pools** > *jdbc_provider*.
  3. Before starting the benchmark workload, right-click *driver* and select **Clear Values**.
  4. (Optional) Right-click driver and select **Reset to Zero**.
  5. Start the workload and run to completion. After the workload finishes, record the value reported for PrepStmt Cache Discards.
  6. Stop the application server and make the following adjustments: Click **Data Source** > **Connection Pooling** > **Statement Cache Size value**.
  7. Rerun the workload and record the Tivoli Performance Viewer value reported for PrepStmt Cache Discards.

  The best value for **Data Source** > **Connection Pooling** > **Statement Cache Size** is the setting used to get either a value of zero or the lowest value for PrepStmt Cache Discards. This setting indicates the most efficient number for a typical workload.
- **Related parameters:** See Adjusting WebSphere Application Server system queues, Thread pool, and Connection pool size

Other JDBC parameters
In addition to setting the prepared statement cache size, you can set other specific properties for JDBC drivers. For example, using Oracle, you can increase the number of rows to fetch while getting result sets with the statement **name=″defaultRowPrefetch″, value=″25″**.

Enter these types of custom properties on the **General** tab for the database.

DB2
DB2 has many parameters that can be configured to optimize database performance. For complete DB2 tuning information, refer to the DB2 UDB Administration Guide: Performance.

DB2 Configuration Advisor: Located in the DB2 Control Center, this advisor calculates and displays recommended values for the DB2 buffer pool size, the

database and database manager configuration parameters, with the option of applying these values. See more information about the advisor in the online help facility within the Control Center.

Use TCP sockets for DB2 on Linux:
- **Short description**: On Linux platforms, whether the DB2 server resides on a local machine with WebSphere Application Server or on a remote machine, configure the DB2 application databases to use TCP sockets for communications with the database.
- **How to view or set**: The directions for configuring DB2 on Linux can be found in the WebSphere Application Server installation documentation for the various operating systems. This document specifies setting DB2COMM for TCP/IP and corresponding changes required in the `etc/service` file.
- **Default value**: Shared memory for local databases
- **Recommended value**: On Linux, change the specification for the DB2 application databases and any session databases from shared memory to TCP sockets.

DB2 MaxAppls:
- **Related parameters**: See Number of connections to DB2

DB2 MaxAgents:
- **Related parameters**: See Number of connections to DB2

DB2 buffpage:
- **Short description**: Buffpage is a database configuration parameter. A buffer pool is a memory storage area where database pages containing table rows or index entries are temporarily read and changed. The purpose of the buffer pool is to improve database system performance. Data can be accessed much faster from memory than from disk.
- **How to view or set**: To view the current value of buffpage for database x, issue the DB2 command **get db cfg for x** and look for the value of BUFFPAGE.

  To set BUFFPAGE to a value of n, issue the DB2 command **update db cfg for x using BUFFPAGE n** and be sure NPAGES is -1 as follows:
  ```
  db2    <-- go to DB2 command mode, otherwise the following "select" will not work as is
  connect to x    <-- (where x is the particular DB2 database name)
  select * from syscat.bufferpools
     (and note the name of the default, perhaps: IBMDEFAULTBP)
     (if NPAGES is already -1, you are OK and no need to issue following command)
  alter bufferpool IBMDEFAULTBP size -1
  (re-issue the above "select" and NPAGES should now be -1)
  ```
- **How to view parameter utilization:** Collect a snapshot of the database while the application is running and calculate the buffer pool hit ratio.
  1. Collect the snapshot by issuing the following DB2 commands:
     - **update monitor switches using bufferpool on**
     - **get monitor switches** (To see that bufferpool monitoring is on)
     - **reset monitor all** (To clear monitor counters)
  2. Run the application.
  3. Issue the following commands:
     - **get snapshot for all databases** (Issue this command before all applications disconnect from the database, otherwise statistics will be lost.)
     - **update monitor switches using bufferpool off** (Make sure not to forget this step!)

4. To calculate the hit ratio, look at the following snapshot statistics for the database:
   - Buffer pool data logical reads
   - Buffer pool data physical reads
   - Buffer pool index logical reads
   - Buffer pool index physical reads
5. The buffer pool hit ratio indicates the percentage of time that the database manager did not need to load a page from disk in order to service a page request. That is, the page was already in the buffer pool. The greater the buffer pool hit ratio, the lower the frequency of disk input/output. The buffer pool hit ratio can be calculated as follows:
   - P = buffer pool data physical reads + buffer pool index physical reads
   - L = buffer pool data logical reads + buffer pool index logical reads
   - Hit ratio = (1-(P/L)) * 100%

DB2 query optimization level:

- **Short description**: When a database query is executed in DB2, various methods are used to calculate the most efficient access plan. The query optimization level parameter sets the amount of work and resources that DB2 puts into optimizing the access plan. The range is from zero to 9.

  An optimization level of 9 causes DB2 to devote a lot of time and all of its available statistics to optimizing the access plan.

- **How to view or set**: The optimization level is set on individual databases and can be set with either the command line or with the DB2 Control Center. Static SQL statements use the optimization level specified on the **prep** and **bind** commands. If the optimization level is not specified, DB2 uses the default optimization as specified by the dft_queryopt parameter. Dynamic SQL statements use the optimization class specified by the current query optimization special register which is set using the SQL Set statement. For example, the following statement sets the optimization class to 1:
  Set current query optimization = 1
  If the current query optimization register has not been set, dynamic statements will be bound using the default query optimization class.

- **Default value**: 5

- **Recommended value**: Set the optimization level for the needs of the application. Use high levels should only when there are very complicated queries.

DB2 reorgchk:

- **Short description**: The performance of the SQL statements can be impaired after many updates, deletes or inserts have been made. Performance can be improved by obtaining the current statistics for the data and rebinding.

- **How to view or set**: Use the following DB2 command to issue runstats on all user and system tables for the database you are currently connected to:
  `reorgchk update statistics on table all`
  You should then rebind packages using the **bind** command.

  In order to see if runstats has been done, issue the following command on DB2 CLP:
  db2 -v "select tbname, nleaf, nlevels, stats_time from sysibm.sysindexes"

  If no runstats has been done, nleaf and nlevels will be filled with -1 and stats_time will have an empty entry "-".

If runstats was done already, the real-time stamp when the runstats was completed will also be displayed under stats_time. If you think the time shown for the previous runstats is too old, do runstats again.

DB2 MinCommit:

- **Short description:** This parameter allows you to delay the writing of log records to a disk until a minimum number of commits have been performed, reducing the database manager overhead associated with writing log records. For example, if MinCommit is set to 2, a second commit would cause output to the transaction log for the first and second commits. The exception occurs when a one-second time out forces the first commit to be output if a second commit does not come along within one second.
- **Actual benefit:** In test applications, up to 90% of the disk input/output was related to the DB2 transaction log. Changing MinCommit from 1 to 2 reduced the results to 45%.
- **When to try adjusting:** Try to adjust this parameter if the disk input/output wait is more than 5% and there is DB2 transaction log activity from multiple sources. When a lot of activity occurs from multiple sources, it is less likely that an single commit will have to wait for another commit (or the one-second time out). Do not adjust this parameter if you have a application with a single thread performing a series of commits (each commit could hit the one-second delay).
- **How to view or set:** To view the current value for a particular database follow these steps:
  1. Issue the DB2 command **get db cfg for xxxxxx** (where xxxxxx is the name of the application database) to list database configuration parameters.
  2. Look for "Group commit count (MINCOMMIT)".
  3. Set a new value by issuing the DB2 command **update db cfg for xxxxxx using mincommit n** (where n is a value between 1 and 25 inclusive).

  The new setting takes effect immediately.

  The following are several metrics that are related to DB2 MinCommit:
  1. The disk input/output wait can be observed on AIX with the command **vmstat 5**. This shows statistics every 5 seconds. Look for the wa column under the CPU area.
  2. The percentage of time a disk is active can be observed on AIX with the command **iostat 5**. This shows statistics every 5 seconds. Look for the %tm_act column.
  3. The DB2 command **get snapshot for db on xxxxxx** (where xxxxxx is the name of the application database) shows counters for log pages read and log pages written.
- **Default value:** The default value is 1.
- **Recommended value:** MinCommit should be 1 or 2 (if the circumstance permits).

Session management

For additional information on setting session management parameters, see the InfoCenter.

Starting Windows NT or 2000 Performance Monitor

From the Start menu, choose **Programs > Administrative Tools > Performance Monitor**.

Additional references

- *WebSphere Application Server Development Best Practices for Performance and Scalability*, which describes development best practices for both Web applications containing servlets, JSP files, and JDBC connections, and enterprise applications containing enterprise bean components.
- iSeries performance documents, including WebSphere Application Server for iSeries Performance Considerations and links to the PTDV tool, Workload Estimator tool, and other documents.
- *IBM WebSphere Application Server Advanced Edition Tuning Guide* (Version 4.02)
- Redbook: *WebSphere Application Server V3.5 Handbook* (SG24-6161-00)
- Redbook: *WebSphere Application Server V3 Performance Tuning Guide* (SG24-5657-00)

Chapter 4. Diagnosing and fixing problems

The purpose of this section is to aid you in understanding why your enterprise application, application server, or WebSphere Application Server is not working and to help you resolve the problem. Unlike performance tuning which focuses on solving problems associated with slow processes and unoptimized performance, problem determination focuses on finding solutions to functional problems.

The kind of problem you are encountering, and how much you already know about it, determine what steps to take to resolve it:

Steps for this task

1. For tips on investigating common problems organized according to tasks within WebSphere Application server, see Troubleshooting by task.
2. For tips on how to investigate common kinds of problems based on the component that is causing the problem, see Troubleshooting by component.
3. If you already have an error message and want to quickly look up its explanation and recommended response, look up the message by selecting the **Quick reference** view of this InfoCenter and expanding **Messages**.
4. For help in knowing where to find error and warning messages, interpreting messages, and configuring log files, see Working with message logs.
5. Difficult problems can require the use of tracing, which exposes the low-level flow of control and interactions between components. For help in understanding and using traces, see Working with trace.
6. For help in adding log and trace capability to your own application, see Programming with JRas.
7. For help in using WebSphere Application Server utilities to help you diagnose the problem, see Working with troubleshooting tools.

   Some of these tools are bundled with the product, and others are freely downloadable.
8. To find out how to look up documented problems, common mistakes, WebSphere Application Server prerequisites, and other problem-determination information on the WebSphere Application Server public web site, or to obtain technical support from IBM, see Obtaining help from IBM.

Troubleshooting by task: what are you trying to do?

This section provides troubleshooting information based on the task you were trying to accomplish when the problem occurred. To find more information about your problem, select a task category from the list below.

If you do not see a task that resembles yours, or if the information provided does not solve your problem, contact IBM support for further assistance.

Installing WebSphere Application Server

Select the problem you are having with WebSphere Application Server installation:
- Install completes with errors or warnings, or installation hangs.
- The installation process completes, but the application server will not start.

- The installation process completes, but sample applications, such as the snoop servlet or other applications from the Sample Gallery do not work.

If you do not see a problem that resembles yours, or if the information provided does not solve your problem, see Troubleshooting the Installation. If that does not provide you with a resolution to your problem, contact IBM support for further assistance

Installation completes with errors or warnings, or hangs (panel appears, but shows no progress)

If the WebSphere Application Server installation program indicates that errors were encountered while installing the product:

- Browse the file main installation log <install_dir/logs/log.txt> for clues.
- Look the command prompt from which the installation panel that hangs was launched, for error messages.
- Look up any error or warning messages in the message reference table.
- For Unix or AIX users, if you have uninstalled WebSphere Application Server prior to reinstalling it, verify that all related packages have been removed by using SMIT or similar tool and looking for packages beginning "WS". If found, remove them.
- Review "Troubleshooting the installation" (not in this document).

If you do not see a problem that resembles yours, or if the information provided does not solve your problem, contact IBM support for further assistance.

Installation completes but the administrative console does not start

What kind of problem are you having?

- "Internal Server Error", "Page cannot be found", 404, or similar error trying to view administrative console.
- "Unable to process login. Please check User ID and password and try again. " error when trying to access console page.
- Directory paths in the console are garbled.

If you are able to bring up the browser page, but the console's behavior is inconsistent, error-prone, or unresponsive, try upgrading the browser you are using. Older browsers may not support the administrative console's features.

If none of these steps solves the problem, check to see if the problem has been identified and documented using the links in Diagnosing and fixing problems: Resources for learning. If you do not see a problem that resembles yours, or if the information provided does not solve your problem, contact IBM support for further assistance.

"**Internal Server Error**", "**Page cannot be found**", **404, or similar error trying to view administrative console:** If you are unable to view the administrative console, here are some steps to try:

- Verify that the application server which supports the administrative console is up and running.
  - For a "base" configuration, the administrative console is deployed by default on "server1". Before viewing the administrative console, you must:
    - Run the **startServer server1** command for Windows or **./startServer.sh server1** command for Unix from a command prompt in the *install_dir*\bin directory, or

- Click the "start application server" link from the "first steps" panel, or
- Start WebSphere Application Server as a service or from the Start menu, if you are using Windows.
  – If you are using the Deployment Manager (for a multi-node configuration), run the startManager command from the *Network_Deployment_install_dir*\bin directory.
  – View the SystemOut.log file for the application server or deployment manager to verify that the server supporting the administrative console has actually started.
- Check the URL you are using to view the console. By default, it is http://*server_name*:9090/admin.
- If you are browsing the console from a remote machine, try to eliminate connection, address and firewall issues by:
  – Pinging the server machine from a command prompt, using the same server name as in the URL.
  – If you have access to the server, try browsing the console locally using http://*server_name*:9090/admin.
- If you have never been able to access the administrative console, verify that the installation was successful.

"**Unable to process login. Please check User ID and password and try again.** " **error when trying to access console page:** This error indicates that security has been enabled for WebSphere Application Server, and the user ID or password supplied is either invalid or not authorized to access the console.

To access the console,
- If you are the administrator, use the ID defined as the security administrative ID. This ID is stored in the WebSphere Application Server directory structure in the file security.xml.
- If you are not the administrator, ask the administrator to enable your ID for the administrative console.

Directory paths in the console are garbled: If directory paths used for classpaths or resources specified in the Application Assembly Tool, configuration files, or elsewhere, appear garbled in the administrative console, it may be because the Java runtime interprets a backslash (\) as denoting a control character.

To resolve, modify Windows-style classpaths by replacing occurrences of single backslashes to two. For example, change "c:\MyFiles\MyJsp.jsp" to "c:\\MyFiles\\MyJsp.jsp".

The application server or Deployment Manager does not start or starts with errors
If the WebSphere Application Server installation program completes successfully, but the application server does not start, or starts with errors:
- Browse the application server log files, which are located by default in *install_dir*\logs*server_name*\SystemErr.log and SystemOut.log for clues.
- If there are several applications deployed on an application server or node, it may take some time to start. Browse the SystemOut.log periodically and look at the most recent updates to see if the server is still starting up. On Unix platforms, the **tail -f** *installation_path***/logs/SystemOut.log** is a convenient way to watch the progress of the server.

- Look for any errors or warnings relating to specific resources with the module, such as Web modules, enterprise beans and messaging resources. If you find any, examine the application server configuration file for that resource's configuration settings. For example, in a base (non-distributed) configuration on Windows systems, browse

  install_dir\config\cells\BaseApplicationServerCell\nodes\host_name\servers\ server_name\server.xml

  and examine the xml tags for that resource's properties. Change its initialState value from "START" to "STOP". Then restart the server as a test to see if the problem is due to this component.
- Look up any error or warning messages in the message reference table by selecting the Quick Reference view and expanding the "Messages" heading.
- If the application server is part of a Network Deployment (multiple server) configuration,
  – Ensure that you have followed the steps for adding the application server to the configuration.
  – Ensure that the configuration is synchronized between the deployment manager and the node. If auto synchronization is running, wait until the synchronization has had a chance to complete. If you are using manual synchronization, request a synchronization to each node in the cluster.
  – Before starting an application server:
    1. Start the Deployment Manager process: *installation_root*/**bin/startManager.sh** or *installation_root***bin\startManager.bat**.
    2. Complete the one-time step of "federating" the node the application server is running on to the Deployment Manager. This has to be done even if there is only one node, and it is the same physical server as the one on which the DeploymentManager is running. This is done by running the addnode *nodename* utility in the *installation_root*/bin directory of the application server's host.
    3. Start the Node Manager process on the nodes hosting the application servers you want to run: *installation_root*/**bin/startNode.sh** or *installation_root***bin\startNode.bat**.
- Ensure that the logical name that you have specified to appear on the console for your application server does not contain invalid characters such as: - / \ : * ? " < >.
- If you are unable to start the DeploymentManager after an otherwise successful installation:
  – Look in the file *installation_root*/dmgr/logs/SystemErr.log and SystemOut.log for messages.
  – Where was the product installed? This product is not standalone, and depends upon some files which are already installed as part of the base. The Network Deployment product should be installed under the WebSphere Application Server root directory of one of the nodes with the base product, at the same level as the base product. For example, if the base product is in /usr/WebSphere/AppServer, the Network Deployment should be installed into a directory like /usr/WebSphere/NetworkDeployment. Installing the product apart from the base product may result in a error running the startManager command similar to: WSVR0102E: An error occurred stopping, null [class com.ibm.ws.cache.ServerCache].

- If you are using Cloudscape and receive an "ERROR XSDB6: Another instance of Cloudscape may have already booted the database databaseName." error starting appplication server, consult this topic for more information.

If you do not see a problem that resembles yours, or if the information provided does not solve your problem, contact IBM support for further assistance.

Installation completes, but sample applications do not work

If the WebSphere Application Server installation program completes successfully, but the sample applications do not run:

- Browse the application server log files, which are located by default in *install_dir*\logs*server_name*\SystemErr.log and SystemOut.log for clues.
- View the JVM logs of the hosting application server for clues, after attempting to run a Sample application,
- Look up any error or warning messages in the message table by selecting the **Quick reference** view of this InfoCenter and expanding the **Messages** heading.
- You can also encounter some security-related problems, such as after turning on security, "MSGS0508E: The JMS Server security service was unable to authenticate userid:" error is displayed in SystemOut.log when starting an application server.
- Review Troubleshooting the installation for more information.

If you do not see a problem that resembles yours, or if the information provided does not solve your problem, check to see if your problem has been identified. See Diagnosing and fixing problems: Resources for learning for information on getting the latest updates. If your problem has not been reported, contact IBM support for further assistance.

Troubleshooting migration problems

To resolve problems encountered in trying to migrate an application from an older version of WebSphere Application Server to version 5, first determine whether your problems occur using the pre-upgrade tool or the post-upgrade tool.

- Errors using the WASPreUpgrade tool.
- Errors using the WASPostUpgrade tool.
- For other kinds of migration problems, such as an application imported from another version of WebSphere Application Server that will not start, look up the related problem under Troubleshooting by task: what are you trying to do?, based on the problem you are having in this version.

If none of these steps fixes your problem,

- For general tips on migration problems, see Troubleshooting the migration utility.
- Review the topic Migration and its subtopics, which address migrating specific kinds of components.
- Check to see if the problem has been identified and documented by looking at the available online support (hints and tips, technotes, and fixes).
- If you don't find your problem listed there contact IBM support.

Errors using the WASPreUpgrade tool

What kind of error are you encountering?

- "MIGR0125E: The call to XMLConfig was not successful"

- "MIGR0108E: The specified WebSphere directory does not contain WebSphere version that can be upgraded."
- "not found" or "no such file or directory" message

Errors using the WASPostUpgrade tool
What kind of error are you encountering?

- "not found" or "no such file or directory" message
- "MIGR0253E: The backup directory migration_backup_directory does not exist"
- "MIGR0102E: Invalid Command Line. MIGR0105E: You must specify the primary node name."
- "MIGR0116E: The backup directory [migration_backup_directory] does not contain the required xml data file."
- "MIGR0108E: The specified WebSphere directory does not contain WebSphere version that can be upgraded"

"**MIGR0125E: The call to XMLConfig was not successful**" error when trying to run WASPreUpgrade
The WASPreUpgrade tool saves selected files from the WebSphere Application Server release 3.5.x and release 4.x `bin` directories. It also exports the existing application server configuration from the repository. If you are migrating WebSphere Application Server Version 3.5.x Advanced Edition or WebSphere Application Server Release 4.x Advanced Edition, the administrative server of the existing environment must be running.

If you are migrating from WebSphere Application Server Release 4.0.x Advanced Edition, the WASPreUpgrade command calls the XMLConfig command to export the existing application server configuration from the repository. If errors occur during this part of the WASPreUpgrade command, you might have to apply fixes to the installation to successfully complete the export step. Contact IBM Support for the latest fixes that might be applicable.

"**MIGR0108E: The specified WebSphere directory does not contain WebSphere version that can be upgraded.**"
Possible reasons for this error follow:

- If WebSphere Application Server Release 4.0.x is installed, you might not have run the WASPreUpgrade tool from the `bin` directory of the version 5 installation root.
  - If you see the following displayed when the WASPreUpgrade tool was run: "IBM WebSphere Application Server, Release 4.0", you are running the WebSphere Application Server Release 4.0 migration utility, not the version 5 migration utility.
  - The resolution is to alter your environment path or change the current directory so that you can launch the WebSphere Application Server version 5 WASPreUpgrade program.
- WebSphere Application Server version 5 might have installed onto the same root directory as the earlier version.
  - Confirm this situation by browsing the older version's directory structure to see whether it contains new 5.0 directories (such as `WebSphere\AppServer\logs\ffdc`).
  - The resolution is to uninstall all versions of WebSphere Application Server, then reinstall and reconfigure the older version, and then install WebSphere Application Server version 5 into a different root directory than the previous one.

- An invalid directory might have been specified when launching the WASPostUpgrade tool, or the WASPreUpgrade tool has not been run.

″**not found**″ or ″**no such file or directory**″ message is returned from the WASPostUpgrade or WASPreUpgrade tool

This problem can occur if you are trying to run the WASPostUpgrade tool or the WASPreUpgrade tool from a directory other than install_dir\bin. The resolution is to ensure that the WASPostUpgrade or WASPreUpgrade .bat or .sh file resides in the install_dir\bin directory, and to launch it from that location.

″**MIGR0253E: The backup directory migration_backup_directory does not exist.**″ error returned from the WASPostUpgrade tool

Possible reasons for this error:

- The WASPreUpgrade tool was not run prior to the WASPostUpgrade tool. To verify this, check to see if the backup directory specified in the error message exists. If not, run the WASPreUpgrade .bat or .sh file, and then retry the WASPostUpgrade tool.

- You might have specified an invalid backup directory. For example, the directory might have been a subdirectory of the V3.5.x or V4.0.x tree, which was deleted after the WASPreUpgrade tool was run and the older version of the product was uninstalled, but before the WASPostUpgrade tool was run.
  - Determine if the full directory structure specified in the error message exists. If possible, rerun the WASPreUpgrade tool, specifying the correct full migration backup directory.
  - If the backup directory does not exist, and the older version it came from is gone, you must rebuild the older version from a backup repository or XML configuration file and rerun the WASPreUpgrade tool.

″**MIGR0102E: Invalid Command Line. MIGR0105E: You must specify the primary node name.**″

The most likely cause of this error is that If release 4.0.x of the WebSphere Application Server is installed, the user might not have run the WASPostUpgrade tool from the bin directory of the WebSphere Application Server verion 5 installation root.

If you received the following messages when the WASPostUpgrade tool was run:

- IBM WebSphere Application Server, Release 4.0 and
-
```
MIGR0002I: java com.ibm.websphere.migration.postupgrade.WASPostUpgrade
        <backupDirectoryName>
        -adminNodeName <primary node name>
        [-nameServiceHost <hostName> [ -nameServicePort <portNumber>]]
        [-substitute <"key1=value1[;key2=value2;[...]]">]
        In input xml file, the key(s) should appear as $key$ for substitution.")
        [-import <xml data file>]
        [-traceString <trace specification> [-traceFile <filename>]]}"
```

  this indicates that the release 4.0 migration tool was run.

To correct this problem, run the WASPostUpgrade command from the bin directory of the WebSphere Application Server version 5 installation root.

″**MIGR0116E: The backup directory [migration_backup_directory] does not contain the required xml data file.**″ error returned from the WASPostUpgrade tool.

Possible reasons for this error:

- If release 4.0.x of WebSphere Application Server is installed, you might not have run the WASPostUpgrade tool from the `bin` directory of the version 5.0 installation root.
    - If "IBM WebSphere Application Server, Release 4.0" is displayed when launching the WASPostUpgrade program, then the wrong version of the program is being executed.
    - To resolve this problem, run the WASPostUpgrade command from the `bin` directory of the 5.0 installation root.

Troubleshooting code deployment and installation problems

Select the problem you are having with deploying or installing developed code for WebSphere Application Server:

- Errors deploying enterprise beans
- Errors or problems deploying, installing, or promoting applications and databases

If you do not see a problem that resembles yours, or if the information provided does not solve your problem, contact IBM support for further assistance.

Errors deploying enterprise beans

What kind of error are you seeing?

- ConnectionFac E J2CA0102E: Invalid EJB component: Cannot use an EJB module with version 1.1 using The Relational Resource Adapter
- WSVR0040E: addEjbModule failed for MyApp-EJB.jar [class com.ibm.ws.runtime.component.DeployedModuleImpl] java.lang.NoClassDefFoundError: com/ibm/ejs/ras/Tr

If none of these errors match the ones you are seeing:

- Browse the application server log files for the server containing the application for clues.
- Look up any error or warning messages in the message table.
- If the application server is part of a Network Deployment (multiple-server) configuration, ensure that you have followed the steps for adding the application server to the configuration.
- If the problems began after WebSphere Application Server security was enabled, view the topic Errors and access problems after enabling security in this InfoCenter.

If you do not see a problem that resembles yours, or if the information provided does not solve your problem, contact IBM support for further assistance.

WSVR0040E: addEjbModule failed for MyApp-EJB.jar [class com.ibm.ws.runtime.component.DeployedModuleImpl] java.lang.NoClassDefFoundError: com/ibm/ejs/ras/Tr: Possible causes of this error include:

- Security permissions are not given for the application in the `<installation_root>\properties\server.policy` file.

  To confirm that this is the problem, check the server.policy file to see if the security permissions are given for application.

  To correct the problem, give permissions for application in the `<strong>server.policy</strong>` file. For example:

```
//purchaseOrder permission
grant codeBase "file:${was.install.root}/installedApps/myApp.ear/-"{
 permission java.security.AllPermission;
 };
```

where myApp.ear is the application name.

For details on how to use the policy tool to configure the server.policy file, see Configuring server.policy files.

- A was.policy file does not exist in the application/META-INF directory, while deploying the application on to the server.

  To confirm that this is the problem, if was.policy exists in application\META-INF directory, then check for syntax errors and make sure the application ear name is given correctly.

  To correct this problem, create a was.policy file in the EAR of the application containing the problem enterprise bean, under the [application]/META-INF directory with the following contents:

```
// WebSphere Application Server Security Policy for the application you are running
grant codeBase "file:myApp.ear" {
permission java.security.AllPermission;
};
```

  where myApp.ear is the application name.

For details on how to use the policy tool to configure the was.policy file, see Configuring was.policy files.

Errors or problems deploying, installing, or promoting applications

What kind of problem are you having?

- I installed my application using wsadmin, but it does not show up under Applications-Manage Applications.
- I get a "java.lang.RuntimeException: Failed_saving_bytes_to_wor_ERROR_" in the Application Assembly Tool (AAT), administrative console or wsadmin
- I get a WASX7015E error running wsadmin command "$AdminApp installInteractive" or "$AdminApp install"..
- A DDL generated by Application Assembly tool throws an SQL error on target platform.
- "ADMA0004E: Validation error in task Specifying the Default Datasource for EJB 1.x Modules" returned when installing application in administrative console or wsadmin.
- "No valid target is specified in ObjectName<object> for module <module>" from installation.
- addNode -includeapps option does not appear to upload all applications to the Deployment Manager.
- "Timeout!!!" error displays when attempting to install an enterprise application in the administrative console.

If none of these steps fixes your problem,

- Ensure that the logical name (the name you have identified to appear on the console) for your application, enterprise bean module or other resource does not contain invalid characters such as these: - / \ : * ? " < > |.

- If the application was installed using the wsadmin `$AdminApp install` command with the `-local` flag, either restart the server or rerun the command without the `-local` flag.

If you do not see a problem that resembles yours, or if the information provided does not solve your problem, check to see if the problem is identified and documented by looking at available online support including hints and tips, technotes, and fixes. If the problem has not been identified, contact IBM support for further assistance.

You may have installed the application but have not saved the configuration afterwards. This can be confirmed by:
- Verifying that the application has its subdirectory under `<install_dir>/installedApps`.
- Running the `$AdminApp list` command and verifying that the application is not among those displayed.
  - In the `bin` directory, run `wsadmin.bat` or `wsadmin.sh`.
  - From the wsadmin prompt, enter `$AdminApp list` and verify that the problem application is not among those displayed.

To resolve, reinstall your application via wsadmin, then run the command `$AdminConfig save` in wsadmin before exiting wsadmin.

″java.lang.RuntimeException: Failed_saving_bytes_to_wor_ERROR_″ in AAT, admin console or wsadmin: If you see this error when attempting to generate deployed code in the AAT, installing an application or module in the administrative console, or using the wsadmin tool to install an application or module, the system's temporary file path length may have been exceeded. This is typically an issue only on Windows platforms.

To verify that this is the problem, check your system's `TEMP` and `TMP` environment variables. If they are long, they are adding path length to the file names accessed by the EJBDeploy tool.

To resolve the problem:
1. Stop all WebSphere Application Server processes and close all DOS prompts.
2. Set the `TMP` and `TEMP` environment variables to something short, for example `C:\TMP` and `C:\TEMP`.
3. Re-install the application.

If this still doesn't work, try rebooting and re-deploy or reinstall the application.

WASX7015E error running wsadmin command ″$AdminApp installInteractive″ or ″$AdminApp install″: This problem has two possible causes:
1. If the full text of the error is similar to:

   ```
   WASX7015E: Exception running command: "$AdminApp installInteractive
   C:/Documents and Settings/myUserName/Desktop/MyApp/myapp.ear";  exception information:
   com.ibm.bsf.BSFException: error while eval'ing Jacl expression: can't find method
   "installInteractive" with 3 argument(s) for class "com.ibm.ws.scripting.AdminAppClient"
   ```

   then the file and path name have been incorrectly specified. In this case, since the path included spaces, it was interpreted as multiple parameters by the wsadmin program.

To resolve this problem, enter the path of the `.ear` file correctly. In this case, by enclosing it in double quotes: `$AdminApp installInteractive "C:\Documents and Settings\myUserName\Desktop\MyApps\myapp.ear"`.

2. If the full text of the error is similar to:

```
WASX7015E: Exception running command: "$AdminApp installInteractive c:\MyApps\myapp.ear ";
exception information: com.ibm.ws.scripting.ScriptingException:  WASX7115E:
Cannot read input file "c:\WebSphere\AppServer\bin\MyAppsmyapp.ear"
```

then the application path is incorrectly specified. In this case, you must use UNIX-style "forward-slash" (/) separators in the path.

DDL generated by Application Assembly tool throws SQL error on target platform: If you receive SQL errors in attempting to execute Data Definition Language statements generated by the Application Assembly Tool on a different platform, for example if you are deploying a CMP enterprise bean designed on Windows onto a Unix server, here are some things to try:

- Browse the DDL statements for dependencies on specific user IDs and passwords, and correct as necessary.
- Browse the DDL statements for dependencies on specific server names, and correct as necessary.
- Refer to the vendor's message reference for causes and suggested actions regarding specific SQL errors. For IBM DB2, these may be viewed online at `http://www.ibm.com/cgi-bin/db2www/data/db2/udb/winos2unix/support/index.d2w/report` .
- If you receive an error similar to

```
SQL0104N  An unexpected token "CREATE TABLE AGENT  (COMM DOUBLE,  PERCENT  DOUBLE,  P"
was found following "        ".  Expected tokens may include:  " ".  SQLSTATE=42601
```

After executing a DDL file created on Windows on a UNIX platform, the problem may be due to a difference in file formats. To resolve this problem:

  – For UNIX platforms other than Linux, edit the DDL in the vi editor, removing the Ctl-M character at the beginning of each line.
  – For Linux, regenerate the deployment code for the application EAR on a Linux platform.

"ADMA0004E: Validation error in task Specifying the Default Datasource for EJB 1.x Modules" returned when installing application in admin console or wsadmin: If you see an error like the following when trying to install an application through the administrative console or the wsadmin command prompt:

```
AppDeploymentException: [ADMA0014E: Validation failed.
ADMA0004E: Validation error in task Specifying the Default Datasource for EJB 1.x Modules
JNDI name is not specified for module beannameBean Jar with URI
filename.jar,META-INF/ejb-jar.xml.
You have not specified the data source for each CMP bean belonging to this module.
Either specify the data source for each CMP beans or specify the default data source
for the entire module.]
```

one possible cause is that in WebSphere Application Server Version 4.0, it was mandatory to have a data source defined for each CMP bean in each JAR. In Version 5, you can specify either a data source for a CMP bean or a default data source for all CMP beans in the JAR. Thus during installation interaction (such as the installation wizard in the Administrative console), the data source fields are optional but the validation performed at the end of the install checks to see at least one of the above is specified.

To correct this problem, step through the installation again, and specify either a default datasource or a datasource for each CMP-type enterprise bean. If you are using the wsadmin tool, either:

- use the `<strong>$AdminApp installInteractive <filename> </strong>` command in order to be prompted for datasources during the installation, or to provide them in a response file.
- specify datasources as an option to the `<strong>$AdminApp install</strong>` command. For details on the syntax, see Installing applications with wsadmin.

"**No valid target is specified in ObjectName <ObjectName>for module <module>**" **from install:** This error can happen in a clustered environment if the target cell, node, server or cluster into which the application is to be installed is incorrectly specified. For example, it can occur if the target is misspelled.

To correct this problem, check the target names against the actual WebSphere Application Server topology and reenter them with corrections.

addNode -includeapps option does not appear to upload all applications to the Deployment Manager: This error can occur when some or all applications on the target node have already been uploaded to the deployment manager. The addNode program detects which applications are already installed and does not upload them again.

To confirm that this is the cause of the problem, use the administrative console to browse the Deployment Manager configuration and see what applications are already installed.

"**Timeout!!!**" **error displays when attempting to install an enterprise application in the administrative console:** This error can happen if you attempt to install an enterprise application that has not been deployed.

To correct this problem:

- Open the `ear` file `<file_name>.ear` in AAT and then select **File ->Generate code for deployment...** This will create a file with a name like `Deployed_<file_name>.ear`.
- In the administrative console, install the deployed `ear` file.

Troubleshooting testing and first time run problems

Select the problem you are having with testing or the first run of deployed code for WebSphere Application Server:

- The application server will not start, or starts with errors.
- The application will not start, or starts with errors.
- A Web resource, such as a JSP, servlet, HTML file, or image, does not display.
- Cannot access a datasource.
- Cannot access an enterprise bean from a servlet, JSP file, standalone program, or other client.
- Cannot access an object hosted by WebSphere Application Server, such as an enterprise bean or connection pool, from a servlet, JSP file, standalone program, or other client.
- I have errors and access problems after enabling security.
- I have errors after enabling Secure Socket Layer (SSL), or SSL-related error messages.

- I have problems with messaging.
- I get errors when trying to send a SOAP request.
- A A WebSphere Application Server Client program does not work.

If you do not see a problem that resembles yours, or if the information provided does not solve your problem, contact IBM support for further assistance.

The application server or Deployment Manager does not start or starts with errors

If the WebSphere Application Server installation program completes successfully, but the application server does not start, or starts with errors:

- Browse the application server log files, which are located by default in *install_dir*\logs*server_name*\SystemErr.log and SystemOut.log for clues.
- If there are several applications deployed on an application server or node, it may take some time to start. Browse the SystemOut.log periodically and look at the most recent updates to see if the server is still starting up. On Unix platforms, the **tail -f** *installation_path*/**logs/**SystemOut.log is a convenient way to watch the progress of the server.
- Look for any errors or warnings relating to specific resources with the module, such as Web modules, enterprise beans and messaging resources. If you find any, examine the application server configuration file for that resource's configuration settings. For example, in a base (non-distributed) configuration on Windows systems, browse install_dir\config\cells\BaseApplicationServerCell\nodes\host_name\servers\server_name\server.xml, and examine the xml tags for that resource's properties. Change its initialState value from "START" to "STOP". Then restart the server as a test to see if the problem is due to this component.
- Look up any error or warning messages in the message reference table by selecting the Quick Reference view and expanding the "Messages" heading.
- If the application server is part of a Network Deployment (multiple server) configuration,
  – Ensure that you have followed the steps for adding the application server to the configuration.
  – Ensure that the configuration is synchronized between the deployment manager and the node. If auto synchronization is running, wait until the synchronization has had a chance to complete. If you are using manual synchronization, request a synchronization to each node in the cluster.
  – Before starting an application server:
    1. Start the Deployment Manager process: *installation_root*/**bin/startManager.sh** or *installation_root***bin****startManager.bat**.
    2. Complete the one-time step of "federating" the node the application server is running on to the Deployment Manager. This has to be done even if there is only one node, and it is the same physical server as the one on which the DeploymentManager is running. This is done by running the addnode *nodename* utility in the *installation_root*/bin directory of the application server's host.
    3. Start the Node Manager process on the nodes hosting the application servers you want to run: *installation_root*/**bin/startNode.sh** or *installation_root***bin****startNode.bat**.
- Ensure that the logical name that you have specified to appear on the console for your application server does not contain invalid characters such as: - / \ : * ? " < >.

- If you are unable to start the DeploymentManager after an otherwise successful installation:
  – Look in the file *installation_root*/dmgr/logs/SystemErr.log and SystemOut.log for messages.
  – Where was the product installed? This product is not standalone, and depends upon some files which are already installed as part of the base. The Network Deployment product should be installed under the WebSphere Application Server root directory of one of the nodes with the base product, at the same level as the base product. For example, if the base product is in /usr/WebSphere/AppServer, the Network Deployment should be installed into a directory like /usr/WebSphere/NetworkDeployment. Installing the product apart from the base product may result in a error running the startManager command similar to: WSVR0102E: An error occurred stopping, null [class com.ibm.ws.cache.ServerCache].
- If you are using Cloudscape and receive an "ERROR XSDB6: Another instance of Cloudscape may have already booted the database databaseName." error starting appplication server, consult this topic for more information.

If you do not see a problem that resembles yours, or if the information provided does not solve your problem, contact IBM support for further assistance.

The application does not start or starts with errors
What kind of error do you see when you start an application?
- java.lang.ClassNotFoundException: <classname>Bean_AdderServiceHome_04f0e027Bean
- ConnectionFac E J2CA0102E: Invalid EJB component: Cannot use an EJB module with version 1.1 using The Relational Resource Adapter
- NMSV0605E: "A Reference object looked up from the context..." error when starting an application.
- other Name Server ("NMSV...") errors.

If none of these erros match the one you see:
- Browse the log files of the application server for this application for clues. By default, these files are: `<install_dir>/logs/<server_name>/SystemErr.log` and `SystemOut.log.`
- Look up any error or warning messages in the message reference table by selecting the Quick Reference view and expanding the "Messages" heading.

If you do not see a problem that resembles yours, or if the information provided does not solve your problem, contact IBM support for further assistance.

java.lang.ClassNotFoundException: <classname>Bean_AdderServiceHome_04f0e027Bean: An exception similar to this happens is you try to start an undeployed application containing enterprise beans, or containing undeployed enterprise bean modules.

enterprise bean modules created in tools like Eclipse or the Application Assembly Tool (AAT) intentionally have incomplete configuration information. Deploying these modules completes the configuration by reading the module's deployment descriptor and completing platform- or installation-dependent settings and adding related classes to the enterprise bean jar file.

To avoid this problem, use one of the following steps:

- Open the undeployed .ear file containing the enterprise bean, or the standalone undeployed .EJB .jar file, in the AAT and run the **File -> Generate code for deployment** option. Then uninstall the application or EJB module in the administrative console and install the deployed version created by the AAT, or
- If you are using the **wsadmin $AdminApp install** command, uninstall it and then reinstall using the -EJBDeploy option. Be sure to follow the install command with the **$AdminConfig save** command.

ConnectionFac E J2CA0102E: Invalid EJB component: Cannot use an EJB module with version 1.1 using The Relational Resource Adapter: This error occurs when an enterprise bean developed to the EJB 1.1 specification is deployed with a WebSphere Application Server V5 J2C-compliant data source, which is the default data source. By default, persistent enterprise beans created under WebSphere Application Server V4.0's Application Assembly tool fulfill the EJB 1.1 specification. In order to run on WebSphere Application Server V5, they must be associated with a WebSphere Application Server V4.0-type data source.

To resolve this problem, you must either modify the application's mapping of enterprise beans to associate 1.x Container Managed Persistence (CMP) beans to associate them with a V4.0 data source or delete the existing data source and create a V4.0 data source with the same name.

To modify the application's mapping of enterprise beans, in the WebSphere Application Server administrative console, select the properties for the problem application and use **map resource references to resources** or **Map data sources for all 1.x CMP beans** to switch the data source the enterprise bean uses, then save the configuration and restart the application.

To delete the existing data source and create a V4.0 data source with the same name:
- In the Administrative Console, select **Resources**->**Manage JDBC Providers**->*JDBC_provider_name*->**Data sources**.
- Delete the data source associated with the EJB 1.1 module.
- Select **Resources**->**Manage JDBC Providers**->*JDBC_provider_name*->**Data sources (Version 4)**.
- Create the data source for the EJB 1.1 module.
- Save the configuration and restart the application.

NMSV0605E: *"A Reference object looked up from the context..."* **error when starting an application:** If the full text of the error is similar to:

```
[7/17/02 15:20:52:093 CDT] 5ae5a5e2 UrlContextHel W NMSV0605E: A Reference object
looked up from the context
    "java:" with the name "comp/PM/WebSphereCMPConnectionFactory" was sent to the
    JNDI Naming Manager
    and an exception resulted. Reference data follows:
    Reference Factory Class Name: com.ibm.ws.naming.util.IndirectJndiLookupObjectFactory
    Reference Factory Class Location URLs:
    Reference Class Name: java.lang.Object
    Type: JndiLookupInfo
    Content: JndiLookupInfo: ; jndiName="eis/jdbc/MyDatasource_CMP"; providerURL="";
    initialContextFactory=""
```

then the problem might be that the data source intended to support a CMP enterprise bean has not been correctly associated with the enterprise bean.

To resolve this problem:

- Select the **Use this Data Source in container managed persistence (CMP)** checkbox in the data source's "General Properties" panel of the administrative console.
- Ensure that the JNDI Name given in Administrative Console under **Resources -> Manage JDBC Provider -> DataSource -> JNDI Name** for DataSource matches the JNDI Name given for CMP or BMP Resource Bindings at the time of Assembling the application in AAT, or
- Check the JNDI Name for CMP or BMP Resource Bindings specified in the code by J2EE Application Developer. One way to do this is to open the deployed .ear folder in the AAT, and look for the JNDI Name for your Entity Beans under CMP or BMP Resource Bindings.

Web resource (JSP file, servlet, HTML file, image) does not display

What kind of error do you see when you start an application?

- Graphics do not appear on jsp or servlet output.
- SRVE0026E: [Servlet Error]-[Unable to compile class for JSP error on JSP.
- After modifying and saving a JSP, the change does not show up in the browser (the old JSP displays).
- Message similar to "Message: /jspname.jsp(9,0) Include: Mandatory attribute page missing"; displays when trying to access JSP.
- The Java source generated from a JSP is not retained in the temp directory (only the classfile is found).
- The JSP Batch Compiler fails with the message "Enterprise Application [application name you typed in] not found."
- Non-English browser input is garbled.
- Scroll bars do not appear around items in the browser window.

Otherwise, if you are not able to display a resource in your browser follow these steps:

- Verify that your HTTP server is healthy by accessing the URL `http://<server_name>` from a browser and seeing whether the "Welcome page" appears. This indicates whether the HTTP server is up and running, regardless of the state of WebSphere Application Server.
- If the HTTP server's "Welcome page" does not appear, that is, you get a browser message such as "page cannot be displayed" or similar, try to diagnose your Web server problem.
- If the HTTP server appears to be functioning, the problem is:
  - The application server may not be serving the target resource. To see if this is the case, try accessing the resource directly through the application server instead of through the HTTP server.
  - If you cannot access the resource directly through the application server:
    - Verify that the URL used to access the resource is correct.
    - If the URL is incorrect and it is created as a link from another JSP file, servlet, or HTML file:
      - After clicking the link, try correcting it by hand in the browser's URL field and reloading, to confirm that the problem is a malformed URL. If this is the problem, correct the URL in the "from" HTML file, servlet or jsp file.

- If the URL appears to be correct, but the resource cannot be accessed directly through the application server, verify the health of the hosting application server and Web module:
  - View the hosting application server and Web module in the administrative console to verify they are up and running.
  - Copy a simple HTML or JSP file (such as `SimpleJsp.jsp`), included in the WebSphere Application Server directory structure) to your Web module's document root, and try to access it. If this works, then the problem is with your resource. View the JVM log of your application server to find out why your resource cannot be found or served
- If the resource can be accessed directly through the application server, but not through an otherwise healthy HTTP server, the problem lies with the HTTP plugin — the component that communicates between the HTTP server and the WebSphere Application Server.
- If JSP and servlet output is served, but not static resources such as `.html` and image files, see the steps for enabling file serving.
- If some kinds of resources display correctly, but you cannot display a servlet by its class name:
  - Ensure that the servlet is in a directory in the Web module's classpath, such as in the `/<Web_module_name>.war/WEB-INF/classes` directory.
  - Ensure that you specify the full class name of the servlet, including its package name, in the URL.
  - Ensure that `"/servlet"` precedes the class name in the URL. For example, example: if the root context of a Web module is `"myapp"`, and the servlet is `com.mycom.welcomeServlet`, then the URL should read:

    `http://<hostname>/myapp/servlet/com.mycom.welcomeServlet`
  - Ensure that serving servlets by classname is enabled for the hosting Web module by opening the source Web module in the Application Assembly Tool and browse the "serve servlets by classname" setting in the IBM Extensions property page. If necessary, enable this flag and redeploy the Web module.
  - For servlets or other resources served by mapped URLs, the URL is `http://<hostname>/<web module context root>/<mappedURL>`.

If none of these steps fixes your problem, check to see if the problem has been identified and documented by looking at the available online support (hints and tips, technotes, and fixes). If you don't find your problem listed there contact IBM support.

Diagnosing Web server problems: If you are unable to view the welcome page of your HTTP server, first determine if the server is operating properly.

On Windows systems, look in the Services panel for the service corresponding to your HTTP server, and verify that the state is "Started". If not, start it. If the service does not start, try starting it manually from the command prompt. If you are using IBM HTTP Server, the command is `<IHS_install_dir>\apache` .

On Unix systems, execute the command `ps -ef | grep httpd`. There should be several processes running with a name of `"httpd"`. If not, start your HTTP server manually. If you are using IBM HTTP Server, the command is `<IHS_install_dir>/bin/apachectl start`.

If the HTTP server will not start:
- Examine the HTTP server's error log for clues.

- Try restoring the HTTP server to its configuration prior to installing WebSphere Application Server and restarting it. If you are using IBM HTTP Server:
  - rename the file `<IHS_install_dir>\httpd.conf`.
  - copy the file `httpd.conf.default` to `httpd.conf`.
  - If Apache is running, stop and restart it.
- For iPlanet, the configuration file to restore is obj.conf. For IIS, try removing the WebSphere Application Server plugin through the IIS administrative GUI.
- If restoring the HTTP server's default configuration works, then manually review the configuration file that has WebSphere Application Server updates to verify the directory and file names for WebSphere Application Server files. If you cannot manually correct the configuration, you may need to uninstall and reinstall WebSphere Application Server in order to create a clean HTTP configuration file.

  If restoring the default configuration file these steps do not help, contact technical support for the Web server you using. If you are using IBM HTTPServer with a WebSphere Application Server purchase, support is included - first check available online support (hints and tips, technotes, and fixes). If you don't find your problem listed there contact IBM support.

Accessing a web resource through the application server (bypassing the HTTP server): Starting with WebSphere Application Server version 4.0, the built-in application server contained in each application server allows you to access Web pages directly, bypassing the HTTP server. It is not recommended to serve a production Web site in this way, but it provides a good diagnostic tool when it is not clear whether a problem resides in the HTTP server, WebSphere Application Server, or the HTTP plugin.

To access a a Web resource through the application server:
- Find out the port of the HTTP service in the target application server.
  - In the WebSphere Administrative Console, select **Servers->Manage Application Servers**.
  - Select the target server, then under Additional Properties select **Web Container**.
  - Under the Additional Properties of the Web Container, select **HTTP Transports**. You will see the ports listed for virtual hosts served by the application server.
  - There may be more than one port listed. In the default application server (server1), for example, 9090 is the port reserved for administrative requests, and 9443 and 9043 are used for SSL-encrypted requests. To simply test the sample ″snoop″ servlet, for example, you would use the default application port 9080, unless it has been changed.
- Using the port number of the Application server's HTTP server, access the resource from a browser. For example, if the port is 9080, the URL would be `http://<hostname>:9080\myAppContext\myJSP.jsp`.
- If you are still unable to access the resource, ensure that the server's HTTP transport port is in the "Host Alias" list:
  1. Select **Application Servers>Your_ApplicationServer>Web Container>HTTP Transports** to check the Default virtual host and the HTTP transport ports used by this application server.

2. Select **Environment>Manage Virtual Hosts>default host>Host Aliases** to check if the HTTP transport port is added. Add an entry if necessary. For example, if the HTTP port for your application is server is 9080, add a host alias of *:9082.

HTTP server and Application Server are working separately, but requests are not passing from HTTP server to Application Server: If your HTTP server appears to be functioning correctly, and the Application Server also works on its own, but browser requests sent to the HTTP server for pages are not being served, this indicates a problem in the WebSphere Application Server plugin.

If this is the case:
- Determine whether the HTTP server is attempting to serve the requested resource itself, rather than forwarding it to the WebSphere Application Server.
    – Browse the HTTP server's access log (`<IHS install root>\logs\access.log` for IBM HTTP Server). It may indicate that it could not find the file in its own document root directory.
    – browse the plugin log file as described below.
- The file `<install_dir>/config/plugin-cfg.xml` determines which requests sent to the HTTP server are forwarded to the WebSphere Application Server, and to which application server. You may need to refresh this file:
    – In the WebSphere Application Server administrative console, expand the Environment tree control.
    – Select **Update WebSphere Plugin**.
    – Stop and restart the HTTP server and retry the Web request.
- Browse the file `<install_dir>/logs/http_plugin.log` for clues to the problem. Make sure the timestamps with the most recent Plugin Information stanza, which is printed out when the plugin is loaded, correspond to the time the Webserver was started.
- Turn on plugin tracing by setting the `LogLevel` attribute in the `<install_dir>/config/plugin-cfg.xml` file to `Trace` and reloading the request, then browsing the `<install_dir>/logs/http_plugin.log` file. You should be able to see the plugin attempting to match the request URI with the various URI definitions for the routes in the `plugin-cfg.xml`. You should be able to see what rules the plugin is not matching against and then figure out if you need to add additional ones. If you just recently installed the application you may need to manually regenerate the plugin configuration in order to pick up the new URIs related to the new application.
- For further details on troubleshooting plugin-related problems, see the topic Troubleshooting the HTTP plugin component.

File serving problems (html, images, etc): If text output appears on your JSP- or servlet-supported Web page, but image files do not:
- Ensure that your files are in the right place: the **document root** directory of your Web application WebSphere Application Server follows the J2EE standard, which means that the document root is the `<Web_module_name>.war` directory of your deployed Web application. Typically this directory will be found in the `<installation_root>/installedApps/<nodename>/<appname>.ear` or `<installation_root>/installedApps/<nodename>/<appname>Network.ear` directory.

    If the files are in a subdirectory of the document root, verify that the reference to the file reflects that. That is, if `invoices.html` is stored in Windows directory

<Web_module_name>.war\invoices, then links from other pages in the Web application to display it should read "invoices\invoices.html", not "invoices.html".

- Ensure that your Web application is configured to enable file serving (i.e., display of static resources like image and .html files):
  - View the file serving property of the hosting Web module by browsing the source .war file in the Application Assembly Tool (AAT). If necessary, update the property and re-deploy the module.
  - Edit the fileServingEnabled property in the deployed Web application's ibm-web-ext.xmi configuration file, typically found in the <install_root>/config/cells/<nodename> or <nodename>Network/applications/<application name>/deployments/<application name>/<Webmodule name>/web-inf directory.

Graphics do not appear on jsp or servlet output: If text output appears on your JSP- or -servlet-supported Web page, but image files do not:

- Ensure that your graphic files are in the right place: the **document root** directory of your Web application WebSphere Application Server 5 follows the J2EE standard, which means that the document root is the <Web_module_name>.war directory of your deployed Web application. Typically this directory will be found in the <installation_root>/installedApps/<nodename>/<appname>.ear or <installation_root>/installedApps/<nodename>/<appname>Network.ear directory.

  If the graphics files are in a subdirectory of the document root, verify that the reference to the graphic reflects that; e.g., if banner.gif is stored in Windows directory <Web_module_name>.war/images, the tag to display it should read: <img SRC="images\banner.gif">, not <img SRC="banner.gif">.

- Ensure that your Web application is configured to enable file serving (i.e., display of static resources like image and .html files).
  - View the file serving property of the hosting Web module by browsing the source .war file in the AAT. If necessary, update the property and re-deploy the module. Or
  - Edit the fileServingEnabled property in the deployed Web application's ibm-web-ext.xmi configuration file, typically found in the <install_root>/config/cells/<nodename> or <nodename>Network/applications/<application name>/deployments/<application name>/<Webmodule name>/web-inf directory.
  - After following one of the above steps:
    - In the administrative console, expand the "Environment" tree control .
    - Click the link "Update WebSphere Plugin" .
    - Stop and restart the HTTP server and retry the Web request.

SRVE0026E: [Servlet Error]-[Unable to compile class for JSP: If this error appears in a browser when trying to access a new or modified .jsp file for the first time, the most likely cause is that the JSP Java source failed (was incorrect) during the javac compilation phase.

To confirm that this is the problem, check the SystemErr.log for a compiler error message, such as:

```
C:\WASROOT\temp\ ... test.war\_myJsp.java:14: Duplicate variable declaration:
int myInt was int myInt
    int myInt = 122; String myString = "number is 122"; static int myStaticInt=22;
    int myInt=121;
            ^
```

If this is the problem, fix the problem in the JSP source, save the source and re-request the JSP.

If this error occurs when trying to serve a JSP that was copied from another system where it ran successfully, then there is something different about the new server's environment that prevents the JSP from running.

Browse the text of the error for a statement like:

```
Undefined variable or class name: MyClass
```

This error indicates that a supporting class or jar file has not been copied to the target server, or is not on the classpath. To resolve, find the file `MyClass.class`, and place it on the Web module's `WEB-INF/classes` directory, or place its containing `.jar` file in the Web module's `WEB-INF/lib` directory.

Verify that the URL used to access the resource is correct:
- For a JSP file, `html` file, or image file:
  **http://<host_name>/<Web_module_context_root>/<subdir under doc root, if any>/<filename.ext> **. The document root for a web application is the `<application_name>.WAR` directory of the installed application.
  - For example, to access `myJsp.jsp`, located in

    c:\WebSphere\ApplicationServer\installedApps\myEntApp.ear\myWebApp.war\invoices

    on `myhost.mydomain.com`, and assuming the context root for the myWebApp Web module is "myApp", the URL would be `http://myhost.mydomain.com/myApp/invoices/myJsp.jsp`.
  - JSP serving is enabled by default. File serving for `html` and image files must be enabled as a property of the Web module, in the Application Assembly Tool, or by setting the **fileServingEnabled** property to "true" in the `ibm-web-ext.xmi`file of the installed Web application and restarting the application.
- For servlets served by class name, the URL is `http://<hostname>/<Web_module_context_root>/servlet/<packageName.className>`.
  - For example, to access myCom.myServlet.class, located in

    c:\WebSphere\ApplicationServer\installedApps\myEntApp.ear\myWebApp.war\WEB-INF\classes

    and assuming the context root for the myWebApp module is "myApp", the URL would be **http://myhost.mydomain.com/myApp/servlet/myCom.MyServlet**.
- Serving servlets by classname must be enabled as a property of the Web module, and is enabled by default. File serving for html and image files must be enabled as a property of the Web application, in the Application Assembly Tool, or by setting the **fileServingEnabled** property to "true" in the **ibm-web-ext.xmi** file of the installed Web application and restarting the application.

Correct the URL in the "from" html file, servlet or jsp: An HREF with no leading "/" inherits the calling resource's context. For example:
- an HREF in `http://[hostname]/myapp/servlet/MyServlet` to "ServletB" resolves to "http://hostname/myapp/servlet/ServletB"

- an HREF in `http://[hostname]/myapp/servlet/MyServlet` to ″servlet/ServletB″ resolves to ″`http://hostname/myapp/servlet/servlet/ServletB`″ (an error)
- an HREF in `http://[hostname]/myapp/servlet/MyServlet` to ″/ServletB″ resolves to ″`http://hostname/ServletB`″ (an error, if `ServletB` requires the same context root as `MyServlet`)

After modifying and saving a JSP, the change does not show up in the browser (the old JSP displays): The most likely cause of this error is that the Web application is not configured for servlet reloading, or the reload interval is too high.

To correct this problem, in the Application Assembly Tool, check the **Reloading Enabled** flag and the **Reload Interval** value in the IBM Extensions for the the Web module in question. Turn Reloading on, or if it is already on then set the Reload Interval lower.

Message like ″**Message: /jspname.jsp(9,0) Include: Mandatory attribute page missing**″ **appears when attempting to browse JSP:** The most likely cause of this error is that the JSP file failed during the translation to Java phase. Specifically, a JSPdirective, in this case an Include statement, was incorrect or referred to a file that could not be found.

To correct this problem, fix the problem in the JSP source, save the source and re-request the JSP.

The Java source generated from a JSP is not retained in the temp directory (only the classfile is found): The most likely cause of this error is that the JSP Processor is not configured to keep generated Java source.

To correct this problem, in the Application Assembly Tool, check the **JSP Attributes** under **Assembly Property Extensions** for the Web module in question. Make sure the attribute **keepgenerated** is there and is set to true. If not, set this attribute and restart the Web application. To see the results of this operation, you will have to delete the classfile from the `temp` directory in order to force the JSP Processor to retranslate the JSP source into Java.

The most likely cause of this error is that the full Enterprise Application path and name, starting with the `.ear` subdirectory that resides in the `<install_root>\config\cells\<node_name>Network\applications` directory is expected as an argument to the JspBatchCompiler tool, not just the display name. For example:
- ″`JspBatchCompiler -enterpriseapp.name sampleApp.ear/deployments/sampleApp`″ is correct, as opposed to
- ″`JspBatchCompiler -enterpriseapp.name sampleApp`″, which is incorrect.

Non-English browser input is garbled: If non-English-character-set browser input is apparently garbled after being read by a servlet or JSP, ensure that the request parameters are encoded according to the expected chararacter set before being read. For example, if the site is Chinese, the target `.jsp` should have a line:
```
req.setCharacterEncoding("gb2312");
```

before any req.getParameter() calls.

Note: This problem especially affects servlets and `jsps` ported from earlier versions of WebSphere Application Server, which converted characters automatically based upon the locale of the WebSphere Application Server.

Scroll bars do not appear around items in the browser window: In some browsers, tree or list type items that extend beyond their allotted windows do not have scroll bars to allow you to see the entire list.

To correct this problem, right click on the browser window and select **Reload** from the pop-up menu.

Cannot access a data source

What kind of database are you trying to access?

- Oracle
- DB2
- SQLServer
- Cloudscape
- Sybase
- my problem wasn't described under the topic for my database, or may not be DBM specific.

If none of these errors match the one you see:

- Browse the application's containing application server's log files for clues. By default these files are `<install_dir/server_name>/SystemErr.log` and `SystemOut.log`.
- Mysterious errors or behavior may be the result of a missing or misnamed Helper Class name. If WebSphere Application Server is not able to load the specified class, it uses a default helper class which may not function correctly with your database manager. Browse the data source's Helper Class property and verify that is correct and is on WebSphere Application Server's classpath.
- Verify that the Java Naming and Directory Interface (JNDI) name of the data source matches the name used by the client attempting to access it. If error messages indicate that the problem may be naming-related, that is they refer to the "name server", "naming service" or include error IDs beginning "NMSV", look at the topics for naming related problems and troubleshooting the Naming Service component.
- Enable tracing for the resource adapter using the trace specification `RRA=all=enabled`. Follow the instructions for dumping and browsing the trace output to narrow the origin of the problem.

If none of these steps fixes your problem, check to see if the problem has been identified and documented by looking at the available online support (hints and tips, technotes, and fixes). If you don't find your problem listed there, contact IBM support.

What kind of error do you see when you try to access your Oracle-based datasource?:

- Invalid Oracle URL specified
- "DSRA0080E: An exception was received by the Data Store Adapter. See original exception message: ORA-00600" when connecting to or using an Oracle data source.
- "Error while trying to retrieve text for error" error when connecting to an Oracle data source.

- java.lang.UnsatisfiedLinkError connecting to an Oracle data source.
- java.lang.NullPointerException or "internal error: oracle.jdbc.oci8.OCIEnv" connecting to an Oracle data source.
- WSVR0016W: Classpath entry, ${ORACLE_JDBC_DRIVER_PATH}/classes12.zip, in Resource, Oracle JDBC Thin Driver, located at cells/BaseApplicationServerCell/nodes/wasrtp/resources.xml has an invalid variable.

What kind of problem are you having accessing your DB2 database?:
- SQL0805N Package "package name" was not found.
- SQLException, with ErrorCode -99,999 and SQLState 58004, with java "StaleConnectionException: COM.ibm.db2.jdbc.DB2Exception: [IBM][CLI Driver] CLI0119E Unexpected system failure. SQLSTATE=58004" using WAS40-type data source.
- DSRA0023E: The DataSource implementation class "COM.ibm.db2.jdbc.DB2XADataSource" could not be found. when trying to access a data source based on a DB2 database.
- SQL0805N Package "NULLID.SQLLC300" was not found. SQLSTATE=51002.
- SQL0567N "DB2ADMIN" is not a valid authorization ID. SQLSTATE=42602.
- CLI0119E System error. SQLSTATE=58004 - DSRA8100 : Unable to get a XAconnection, or DSRA0011E: Exception: COM.ibm.db2.jdbc.DB2Exception: [IBM][CLI Driver] CLI0119E Unexpected system failure. SQLSTATE=58004.
- COM.ibm.db2.jdbc.DB2Exception: [IBM][CLI Driver][DB2/NT] SQL0911N The current transaction has been rolled back because of a deadlock or timeout. Reason code "2". SQLSTATE=40001.
- (Unix)java.sql.SQLException: java.lang.UnsatisfiedLinkError: Can't find library db2jdbc (libdb2jdbc.a or .so) in java.library.path.
- "COM.ibm.db2.jdbc.DB2ConnectionPoolDataSource" could not be found for data source (data_source).

What kind of problem are you having accessing your SQLServer database?:
- ERROR CODE: 20001 and SQL STATE: HY000.
- Application fails with message stating "Cannot find stored procedure..."

What kind of problem are you having accessing your Cloudscape database?:
- Unexpected IOException wrapped in SQLException, accessing Cloudscape database.
- "Select for update" on one row causes table to become locked, triggering a deadlock condition.
- "ERROR XSDB6: Another instance of Cloudscape may have already booted the database databaseName." error starting appplication server.

Note: Cloudscape errorCodes (2000, 3000, 4000) indicate levels of severity, not specific error conditions. In diagnosing Cloudscape problems, pay attention to the given sqlState value.

What kind of problem are you having accessing your Sybase database?:
- SET CHAINED command not allowed within multi-statement transaction.
- "Sybase Error 7713: Stored Procedure can only be executed in unchained transaction mode" error.
- "JZ0XS: The server does not support XA-style transactions. Please verify that the transaction feature is enabled and licensed on this server."

- A Container Managed Persistence (CMP) enterprise bean is causing exceptions.

What kind of general data access problem do you have?:
- "ObjectNotFoundException", "NameNotFoundException", or other jndi-related error when the client application attempts to use the data source.
- "IllegalConnectionUseException"
- WTRN0062E: An illegal attempt to enlist multiple one phase capable resources has occurred.
- ConnectionWaitTimeoutException.
- com.ibm.websphere.ce.cm.StaleConnectionException: [IBM][CLI Driver] SQL1013N The database alias name or database name "NULL" could not be found. SQLSTATE=42705
- java.sql.SQLException: java.lang.UnsatisfiedLinkError:
- "J2CA0030E: Method enlist caught java.lang.IllegalStateException" wrapped in error "WTRN0063E: An illegal attempt to enlist a one phase capable resource with existing two phase capable resources has occurred" when attempting to execute a transaction.

Invalid Oracle URL specified: This error may be caused by an incorrectly specified URL on the target data source's URL property.
- Examine the URL property for the data source object in the administrative console. For the 8i OCI driver, make sure oci8 is used in URL. For the 9i OCI driver, you can use either oci8 or oci.
- Examples of Oracle URLs:
  - For the thin driver: jdbc:oracle:thin:@hostname.rchland.ibm.com:1521:IBM
  - For the thick (OCI) driver: jdbc:oracle:oci8:@tnsname1

"**DSRA0080E: An exception was received by the Data Store Adapter. See original exception message: ORA-00600**" **when connecting to or using an Oracle data source** "**DSRA0080E: An exception was received by the Data Store Adapter. See original exception message: ORA-00600**" **when connecting to or using an Oracle data source** "**DSRA0080E: An exception was received by the Data Store Adapter. See original exception message: ORA-00600**" **when connecting to or using an Oracle data source:** A possible reason for this exception is that a version of the Oracle JDBC driver is being used that is older than the Oracle database that is being connected to, or more than one version of the Oracle JDBC driver has been configured on the WebSphere Application Server.

To confirm that this is the problem, examine the version of the JDBC driver that is being used. This can sometimes be determined by looking at the classpath to determine what directory the driver is in. If you can't determine the version this way, you can use the following program to determine the version.

Before running this program, set the classpath to the location of the JDBC driver files.

```
import java.sql.*;
    import oracle.jdbc.driver.*;
    class JDBCVersion
    {
        public static void main (String args[])
        throws SQLException
        {
            // Load the Oracle JDBC driver
            DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
```

```
            // Get a connection to a database
            Connection conn = DriverManager.getConnection
                    ("jdbc:oracle:thin:@appaloosa:1521:app1","sys","change_on_install");
            // Create Oracle DatabaseMetaData object
            DatabaseMetaData meta = conn.getMetaData();
            // gets driver info:
            System.out.println("JDBC driver version is " + meta.getDriverVersion());
        }
    }
```

If this proves to be the problem, simply replace the JDBC driver with the correct version, or if multiple drivers are configured, remove the incorrect one.

″**Error while trying to retrieve text for error**″ **error when connecting to an Oracle data source:** The most likely cause of this error is that the Oracle 8i OCI driver is being used and the ORACLE_HOME property is either not set or is set incorrectly.

To correct this error, examine the user profile that WebSphere Application Server is running under and verify that it has the $ORACLE_HOME environment variable set correctly.

″**java.lang.UnsatisfiedLinkError:**″ **connecting to an Oracle data source:** If your data source throws an **UnsatisfiedLinkError**, and the full exception indicates that the problem is related to an Oracle module, as in the following example for the 8i driver:

```
Exception in thread "main" java.lang.UnsatisfiedLinkError:
    /usr/WebSphere/AppServer/java/jre/bin/libocijdbc8.so:  load ENOENT on shared library(s)
    /usr/WebSphere/AppServer/java/jre/bin/libocijdbc8.so libclntsh.a
```

or in this example for the 9i driver:

```
Exception in thread "main" java.lang.UnsatisfiedLinkError:
    no ocijdbc9  (libocijdbc9.a or .so) in java.library.path
        at java.lang.ClassLoader.loadLibrary(ClassLoader.java(Compiled Code))
        at java.lang.Runtime.loadLibrary0(Runtime.java:780)
```

the problem may be that the environment variable LIBPATH is not set or is set incorrectly.

To correct this problem, examine the user profile that WebSphere Application Server is running under and correct the LIBPATH environment variable if necessary to include Oracle libraries. Scan for the file ″lobocijdbc8.so″ in order to find the right directory.

java.lang.NullPointerException referencing 8i classes, or ″ **internal error: oracle.jdbc.oci8. OCIEnv**″ **connecting to an Oracle data source:** If you encounter an exception similar to either of the following when your application attempts to connect to an Oracle data source:

```
Exception in thread "main" java.lang.NullPointerException
        at oracle.jdbc.oci8.OCIDBAccess.check_error(OCIDBAccess.java:1743)
        at oracle.jdbc.oci8.OCIEnv.getEnvHandle(OCIEnv.java:69)
        at oracle.jdbc.oci8.OCIDBAccess.logon(OCIDBAccess.java:452)
        at oracle.jdbc.driver.OracleConnection. <init>(OracleConnection.java:287)
```

or

```
Exception in thread "main" java.sql.SQLException:
    internal error: oracle.jdbc.oci8. OCIEnv@568b1d21
        at oracle.jdbc.dbaccess.DBError.throwSqlException(DBError.java:184)
        at oracle.jdbc.dbaccess.DBError.throwSqlException(DBError.java:226)
        at oracle.jdbc.oci8.OCIEnv.getEnvHandle(OCIEnv.java:79)
```

this indicates that the 9i OCI driver is being used on AIX 32 bit machine, and the LIBPATH is set correctly, but ORACLE_HOME is not set or is set incorrectly.

To correct this problem, examine the user profile that WebSphere Application Server is running under and verify that it has the $ORACLE_HOME environment variable set correctly, and that $LIBPATH includes $ORACLE_HOME/lib .

WSVR0016W: Classpath entry, ${ORACLE_JDBC_DRIVER_PATH}/classes12.zip, in Resource, Oracle JDBC Thin Driver, located at cells/BaseApplicationServerCell/nodes/wasrtp/resources.xml has an invalid variable: This error occurs when no environment variable is defined for the property: `ORACLE_JDBC_DRIVER_PATH` .

To verify that this is the problem, go to the administrative console in the left panel under **Environment ->Manage WebSphere Variables** check whether the variable `ORACLE_JDBC_DRIVER_PATH` is defined.

To correct the problem,

1. In this panel, click on **New** and define the variable. For example, name : `ORACLE_JDBC_DRIVER_PATH` , value : `c:\oracle\jdbc\lib` (the actual value depends on your operating system and directory structure, but it should be the directory containing the `classes12.zip` file).

SQL0805N Package <package_name> was not found: Possible reasons for these exceptions are:

- If the package name is `NULLID.SQLLC300`, see SQL0805N Package "NULLID.SQLLC300" was not found. SQLSTATE=51002.
- You are is attempting to use a XA-enabled JDBC driver on a DB2 database that isn't XA-ready. To correct this problem:
  – If the database is Db2/UDB, run the following commands (using the db2cmd interface) while connected to the database in question:
    - **DB2 bind @db2ubind.lst blocking all grant public**
    - **DB2 bind @db2cli.lst blocking all grant public**

    Note: The files `db2ubind.lst` and `db2cli.lst` are in the "bnd" directory of your DB2 install. It is suggested you run these commands from that directory. You should only need to do this once.

SQLException, with ErrorCode -99,999 and SQLState 58004, with java "StaleConnectionException: COM.ibm.db2.jdbc.DB2Exception: [IBM][CLI Driver] CLI0119E Unexpected system failure. SQLSTATE=58004", when using WAS40-type data source: "Unexpected system failure" usually occurs when running in XA mode (two-phase commit). There are many possible reasons, including:

- An invalid username or password was provided.
  – To determine whether you have a username/password problem, look in db2diag.log and view the actual error message and sql code. A message like the following, with an SQLCODE of -1403, indicates an invalid user ID or password:

```
2002-07-26-14.19.32.762905   Instance:db2inst1   Node:000
        PID:9086(java)   Appid:*LOCAL.db2inst1.020726191932
        XA DTP Support  sqlxa_open   Probe:101
```

```
DIA4701E Database "POLICY2" could not be opened for distributed transaction
processing.
String Title: XA Interface SQLCA  PID:9086 Node:000
SQLCODE = -1403
```

- The database name is incorrect.
- Some db2 packages are corrupted.

To resolve these problems:

- Correct your username and password, if you specified your password on the GUI (for 40 Datasource) make sure that the username and password specified on the bean itself are correct, since they will overwrite whatever you specify when creating the data source.
- Use the correct database name.
- Rebind the packages (which are found in the bnd directory) as follows:

```
db2connect to dbname
c:\SQLLIB\bnd>DB2 bind @db2ubind.lst blocking all grant public
        c:\SQLLIB\bnd>DB2 bind @db2cli.lst blocking all grant public
```

- Ensure that the file \WebSphere\AppServer\properties\wsj2cdpm.properties has the right userid/password.

Error message ″**java.lang.reflect.InvocationTargetException: com.ibm.ws.exception.WsException: DSRA0023E: The DataSource implementation class** ″**COM.ibm.db2.jdbc.DB2XADataSource**″ **could not be found.**″ **when trying to access a DB2 database:** One possible reason for this exception is that a user is attempting to use a JDBC 2.0 DataSource, but DB2 isn't JDBC 2.0 enabled. This frequently happens with new installations of DB2 because DB2 provides separate drivers for JDBC 1.X and 2.0, with the same physical file name, and by default the 1.X driver is placed on the classpath.

To confirm that this is the problem:

- On Windows systems, look for the file **inuse** in the java12 directory under your DB2 install. If it is not there, you need to run usejdbc2.bat.
- On Unix systems, check the classpath for your data source. If it doesn't point to the db2java.zip under the java12 directory, you will need to change it.

To correct this problem:

- On Windows systems, stop DB2 and run usejdbc2.bat from the java12 directory in your DB2 installation. It is suggested you run this from the command line to make sure it completes successfully.
- On Unix systems, change the classpath for your data source to point to the db2java.zip in the java12 directory of your DB2 installation.

SQL0805N Package ″**NULLID.SQLLC300**″ **was not found. SQLSTATE=51002:** Some possible causes of this error are:

- The underlying database was dropped and recreated.
- DB2 was ugpraded, and its packages may not have been rebound correctly.

To resolve this problem, rebind the db2 packages by running the the db2cli.lst script found in the bnd directory. For example:db2>@db2cli.lst .

SQL0567N ″**DB2ADMIN** ″ **is not a valid authorization ID. SQLSTATE=42602:** If you encounter this error when attempting to access a Db2/UDB data source:

- Check your username and password in the data source properties in the admin console. Ensure that they are correct.

- Ensure that the userid and password do not contain blank characters (before, in between or after).

CLI0119E System error. SQLSTATE=58004 - DSRA8100 : Unable to get a XAconnection or DSRA0011E: Exception: COM.ibm.db2.jdbc.DB2Exception: [IBM][CLI Driver] CLI0119E Unexpected system failure. SQLSTATE=5800: If you encounter this error when attempting to access a DB2/UDB data source:

- Check your username and password "custom properties" in the data source properties page in the admin console. Ensure that they are correct.
- Ensure the userid and password do not containe any blank characters (before, in between or after).
- Check that the WAS.policy file exists for the application, for example, D:\WebSphere\AppServer\installedApps\markSection.ear\META-INF\was.policy.
- View the entire exception listing for an underlying SQL error, and look it up using the dbm vendor's message reference.

If you encounter this error while running DB2 on Red Hat Linux, it indicates that the max queues system wide parameter is set too low to allow DB2 to acquire the necessary resources to complete the transaction. When this is the problem, exception DSRA8100E can be preceded by exceptions J2CA0046E and DSRA0010E.

To correct this problem, edit the file /proc/sys/kernal/msgmni and increase the value of the max queues system wide parameter to be greater than 128.

COM.ibm.db2.jdbc.DB2Exception: [IBM][CLI Driver][DB2/NT] SQL0911N The current transaction has been rolled back because of a deadlock or timeout. Reason code ″2″. SQLSTATE=40001: If you see an error similar to the following when accessing a Db2 data source:

```
ERROR CODE: -911
COM.ibm.db2.jdbc.DB2Exception: [IBM][CLI Driver][DB2/NT] SQL0911N
The current transaction has been rolled back because of a deadlock
or timeout.  Reason code "2".  SQLSTATE=40001
```

the cause is probably an application-caused Db2 deadlock.

To diagnose the problem:
- Execute the Db2 commands:
  - db2 update monitor switches using LOCK ON
  - db2 get snapshot for LOCKS on dbName ><directory_name>\lock_snapshot.log

  where d:\lock_snapshot.log now has the DB2 lock information.
- Turn off the lock monitor by executing: db2 update monitor switches using LOCK OFF.
- to see if you got a deadlock:
  - Look for an application handle that has a lock-wait status, then look for the "ID of agent holding lock"
  - Go to that handle. If that handle has a lock-wait status and the "ID of agent holding lock" for that it is the previous one, then you know that you have circular lock (deadlock).

To resolve this problem:

- Examine your application and use a less restrictive isolation level if no concurrency access is needed.
- Use caution; moving to a lesser **accessIntent** could result in data integrity problems.

"**COM.ibm.db2.jdbc.DB2ConnectionPoolDataSource**" **could not be found for data source ([data-source-name])**":   This error usually occurs when the classpath of the DB2 jdbc driver is set correctly to "${DB2_JDBC_DRIVER_PATH}/db2java.zip" but the environment variable "DB2_JDBC_DRIVER_PATH" has not been set.

To confirm that this is the problem, on the **Manage WebSphere Variables** panel, under **Environment**, and verify that there is no entry for the variable "DB2_JDBC_DRIVER_PATH".

To correct this problem, add the variable DB2_JDBC_DRIVER_PATH, with the value set to the directory path containing db2java.zip.

ERROR CODE: 20001 and SQL STATE: HY000 accessing SQLServer database: If you see an error similar to the following when attempting to access an SQLServer database:

```
ERROR CODE: 20001
SQL STATE:  HY000
java.sql.SQLException: [Microsoft][SQLServer JDBC Driver][SQLServer]xa_open (0) returns -3
at com.microsoft.jdbc.base.BaseExceptions.createException(Unknown Source) ...
at com.microsoft.jdbcx.sqlserver.SQLServerDataSource.getXAConnection(Unknown Source) ...
```

the problem may be that the Distributed Transaction Coordinator service is not started.

To confirm that this is the problem, in the Windows **Control Panel -> Services** window, check whether the service "Distributed Transaction Coordinator" or "DTC" is started. If not, it may be the cause of the problem.

To resolve this problem, start the Distributed Transaction Coordinator.

Application fails with message stating "**Cannot find stored procedure...**" **accessing an SQLServer database:**   One possible cause for this error is that the Stored Procedures for JTA feature was not installed on the Microsoft SQL Server.

To correct this problem, repeat the installation for Stored Procedures for JTA according to the ConnectJDBC installation guide.

Unexpected IOException wrapped in SQLException, accessing Cloudscape database: This problem can occur because Cloudscape databases use a large number of files, Some operating systems, such as Sun Solaris, limit the number of files an application can open at one time. If the default is a low number, such as 64, you may get this exception.

If your operating system lets you configure the number of file descriptors, you can correct this problem by setting the number to a high value, such as 1024.

"**select for update**" **causes table lock and deadlock when accessing Cloudscape:** If executing a "select for update" on one row causes the entire table is to be locked, which in turn creates a deadlock condition, the cause may be that you don't have indexes defined on that table, particularly on the columns you use in the **where** clause, since Cloudscape creates a table lock rather than a row level lock.

To resolve this problem, create an index on the affected table.

ERROR XSDB6: Another instance of Cloudscape may have already booted the database ″**database**″**:** This problem is caused by the fact that Cloudscape 5.0.X allows only one JVM to have access to database instance at a time.

To resolve this problem:
- Ensure that you don't have other jdbc client programs, such as **ij** or **cview** running on that database instance when WebSphere Application Server is running.
- Ensure that you don't use the same instance as the database for more than one data source.

″**SET CHAINED command not allowed within multi-statement transaction.**″ **exception accessing Sybase:** If you see an error similar to the following when attempting to use a Sybase data source:

```
[7/30/02 9:44:06:191 CDT] 3ab306e5 SybaseDataSto d The sqlState is: ZZZZZ
  [7/30/02 9:44:06:191 CDT] 3ab306e5 GenericDataSt > findMappingClass for exception
  com.sybase.jdbc2.jdbc.SybSQLException: SET CHAINED command not allowed within
  multi-statement transaction.
```

the cause may be:
- You are attempting to set autocommit to "on" in a 2 phase transaction, which is not permitted. To resolve this problem either:
  - do not modify the autocommit value, or
  - use a single phase data source.
- You might have an incorrectly configured DSM license. To resolve this problem, correct the **Adaptive Server Enterprise DTM** option **Authorization Code**. This is the license code supplied by your Sybase dealer. It can be entered into the license.dat file in the Sybase directory structure.

″**Sybase Error 7713: Stored Procedure can only be executed in unchained transaction mode**″ **error:** This error occurs when the JDBC attempts to put the connection in `autocommit(true)` mode. The application can change the connection to chained mode using `Connection.setAutoCommit(false)`, or by using a ″set chained on″ language command. This error is caused when the stored procedure was not created in a compatible mode.

To fix this problem, use: `sp_procxmode <procedure_name>`**,** ″anymode″.

″**JZ0XS: The server does not support XA-style transactions. Please verify that the transaction feature is enabled and licensed on this server.**″**:** This error occurs when XA-style transactions are attempted on a server that does not have Distributed Transaction Management (DTM) installed.

To correct this problem, use the instructions in the Sybase Manual titled: *Using Adaptive Server Distributed Transaction Management Features* to enable Distributed Transaction Management (DTM). The main steps in this procedure are:
1. Install the DTM option.
2. Check the `license.dat` file to verify that the DTM option was installed.
3. Restart the license manager.
4. Enable DTM in ISQL.
5. Restart the ASE service.

A Container Managed Persistence (CMP) enterprise bean is causing exceptions:
This error is caused by improper use of reserved words. Reserved words cannot be
used as column names.

To correct this problem, rename the variable to remove the reserved word. You can
find a list of reserved words in the *Sybase Adaptive Server Enterprise Reference
Manual; Volume 1: Building Blocks*, Chapter 4. This manual is available online at:
http://manuals.sybase.com/onlinebooks/group-as/asg1250e/refman.

IllegalConnectionUseException: One possible reason for this error is that a
connection obtained from a WAS40DataSource is being used on more than one
thread. This is a violation of the J2EE 1.3 programming model, and an exception is
generated when it is detected on the server. This problem occurs for users
accessing a data source through servlets or Bean Managed Persistence (BMP) type
enterprise beans.

To confirm that this is the problem, examine the code for sharing of connections.
Code can inadvertently cause sharing by not following the programming model
recommendations, for example by storing a connection in an instance variable in a
servlet, which can cause the connection to be used on multiple threads at the same
time.

**WTRN0062E: An illegal attempt to enlist multiple one phase capable resources
has occurred:** Possible causes of this error include:
- An attempt to have a one phase resource participate in a global transaction
  while an XA resource or another one phase resource has already participated in
  this global transaction.
  - Within the scope of a global transaction you've tried to get a connection more
    than once and at least one of the resource-refs you are using specifies that the
    connection is unshareable, and the data source is not configured to support 2
    Phase Commit transactions. That is, it does not support an XAResource. If
    you are not using a resource-ref you will also default to unshareable
    connections.
  - Within the scope of a global transaction you've tried to get a connection more
    than once and at least one of the resource-refs you are using specifies that the
    connection is shareable and the data source is not configured to support 2
    Phase Commit transactions. That is, it does not support an XAResource. In
    addition, even though you have specified that connections should be
    shareable, each getConnection request was made with different connection
    properties (such as IsolationLevel or AccessIntent). In this case the
    connections are not shareable, and multiple connections will be handed back.
  - Multiple components (Servlets, Session Beans, BMP Entity Beans, or CMP
    Entity Beans) are being accessed within a global transaction. All use the same
    DataSource and all specify shareable connections on their resource-ref's and
    you expect them to all share the same connection. If the properties are
    different, as stated above, you will get multiple connections. AccessIntent
    settings on CMP beans will change their properties. In order for them to share
    a connection the AccessIntent setting must be the same. If you want CMP
    beans to share a connection with non-CMP components, see the related
    documentation in the DataAccess section of the InfoCenter.
- An attempt to have more than one unshareable connections participate in a
  global transaction, when the data source is not an XA resource.
- An attempt to share a single phase connection, when each getConnection
  method has different connection properties; such as the AccessIntent. This causes
  the connection to be created as non-shareable.

To correct this error:

- Check the jdbc provider **implementation** class from the **Manage JDBC resource** panel of the administrative console to ensure that it is a class that supports XA-type transactions.
- Check the connection's sharing scope from the resource binding, using the AAT.
  - If you are running a unshareable connection scope, then ensure that your data source is an XA data source.
  - If you are running a shareable connection scope, then ensure that all the connection properties, such as AccessIntent and any other properties (such as userid), are the same.
- Check what your client code passes in with its getConnection requests, and make sure they are consistent with each other.

ConnectionWaitTimeoutException accessing a data source or resource adapter: If your application receives a com.ibm.websphere.ce.cm.ConnectionWaitTimeoutException or com.ibm.websphere.ce.j2c.ConnectionWaitTimeoutException when attempting to access a WebSphere Application Server data source or JCA-compliant resource adapter, respectively, some possible causes are:

- The maximum number of connections for a given pool is set too low. The demand for concurent use of connections is greater then the configured maximum for the connection pool. One indication that this is the problem is that you receive these exceptions regularly, but your CPU utilization is not high. This indicates that there are too few connections available to keep the threads in the server busy.
- Connection Wait Time is set too low. Current demand for connections is high enough such that sometimes there is not an available connection for short periods of time. If your connection wait timeout value is too low, you may timeout shortly before a user returns a connection back to the pool. Adjusting the connection wait time may give you some relief. One indication that this is the problem is that you are using near the maximum number of connections for an extended period and receiving this error regularly.
- You are not closing some connections or are returning connections back to the pool at a very slow rate. This can easily happen when using unshareable connections, when you forget to close them, or you close them long after you are finished using them, thus keeping the connection from being returned to the pool for reuse. The pool soon becomes empty and all applications get ConnectionWaitTimeoutExceptions. One indication that this is the problem is that the connection pool has become starved and you receive this error on most requests.
- You are driving more load than the server or backend system have resources to handle. In this case you must determine which resources you need more of and upgrade configurations or hardware to address the need. One indication that this is the problem is that the application or database server CPU is nearly 100% busy.

To correct these problems, modify an application to use fewer connections or properly close the connections, change the pool settings of MaxConnections or ConnnectionWaitTimeout, or adjust resources and their configuration.

com.ibm.websphere.ce.cm.StaleConnectionException: [IBM][CLI Driver] SQL1013N The database alias name or database name ″NULL″ could not be

found. SQLSTATE=42705: This error occurs when a data source has been defined but the databaseName attribute and corresponding value have not been added to the "custom properties".

To add the the databaseName property:

1. Expand the **Resources->Manage JDBC Providers** link in the administrative console.
2. Select the JDBC Provider which supports the problem data source.
3. Select **Data Sources** and then select the problem data source.
4. Under **additional properties** select **Custom Properties**.
5. Select the **databaseName** property, or add one if it does not exist, and enter the actual database name as the value.
6. Click **Apply** or **OK**, and then **Save** from the action bar.
7. Try to access the data source again.

java.sql.SQLException: java.lang.UnsatisfiedLinkError:: This error indicates that the directory containing the binary libraries which support a database are not included in the LIBPATH environment variable for the environment in which the WebSphere Application Server is started.

The path containing the DBM vendor's libraries vary by dbm. One way to find them is by scanning the for missing library specified in the error message. Then the LIBPATH variable can be corrected to include the missing directory, either in the `.profile` of the account from which WebSphere Application Server is executed, or by adding a statement in a `.sh` file which then executes the "startServer" program.

"J2CA0030E: Method enlist caught java.lang.IllegalStateException" wrapped in error "WTRN0063E: An illegal attempt to enlist a one phase capable resource with existing two phase capable resources has occurred" when attempting to execute a transaction.: This error can occur when Last Participant Support (LPS) is missing or disabled. LPS allows a one-phase capable resource and a two-phase capable resource to be enlisted within the same transaction.

LPS is only available if the following are true:
- WebSphere Application Server Programming Model Extensions (PME), which is included in the Application Server Enterprise product) is installed.
- The option "Additional Enterprise Extensions" was enabled when PME was installed. If you perform a typical installation, this is be enabled by default. If you perform a custom installation, you have the option to disable this function, which would disable LPS.
- The application enlisting the one phase resource has been deployed with the **Accept heuristic hazard** option enabled. This is done through the Application Assembly Tool. To enable this option in the Application Assembly Tool:
  1. Load the EAR file into the Application Assembly Tool.
  2. If the EAR file is actually a JTEE1.2 EAR then it must be upgraded to a JTEE1.3 EAR by selecting File-> Convert EAR from the Application Assembly Tool.
  3. Select the EAR file in the left-hand panel of the Application Assembly Tool.
  4. Select the **WAS Enterprise** tab in bottom right-hand window panel of the Application Assembly Tool.
  5. Ensure that the **Accept heuristic hazard** option is selected.

6. Save the EAR file.

Cannot access an enterprise bean from a servlet, JSP file, standalone program, or other client

What kind of error are you seeing?

- javax.naming.NameNotFoundException: Name name not found in context "local" message when access is attempted
- BeanNotReentrantException is thrown
- CSITransactionRolledbackException / TransactionRolledbackException is thrown
- Call fails, stack trace beginning "EJSContainer E Bean method threw exception [exception_name]" found in JVM log file.
- Call fails, ObjectNotFoundException or ObjectNotFoundLocalException when accessing stateful session EJB found in JVM log file.
- Attempt to start CMP EJB module fails with javax.naming.NameNotFoundException: dataSourceName
- Transaction [tran ID] has timed out after 120 seconds error accessing EJB.
- Symptom:CNTR0001W: A Stateful SessionBean could not be passivated

If the client is remote to the enterprise bean, that is running in a different application server or as a standalone client, browse the JVM logs of the application server hosting the enterprise bean as well as log files of the client.

If you do not see a problem that resembles yours, or if the information provided does not solve your problem,

- If the problem appears to be Name-Service related, that is, you see a NameNotFoundException, or a message ID beginning NMSV
  - See Cannot access an object hosted by WebSphere Application Server from a servlet, jsp, or other client and Naming services component troubleshooting tips for more information.
- Check to see if the problem has been identified and documented using the links in Diagnosing and fixing problems: Resources for learning.
- If none of these fixes your problem, contact IBM support for further assistance.

ObjectNotFoundException or ObjectNotFoundLocalException when accessing stateful session EJB: A possible cause of this problem is that the stateful session bean timed out and was removed by the container. This event must be coded for, according to the EJB 2.0 specification (available at `http://java.sun.com/products/ejb/docs.html`), section 7.6.2, Dealing with exceptions.

Stack trace beginning "EJSContainer E Bean method threw exception [exception_name]" found in JVM log file: If the "exception name" indicates an exception thrown by an IBM class, that is it begins "com.ibm...", then search for the exception name within this InfoCenter, and in the online help as described below. If "exception name" indicates an exception thrown by your application, contact the application developer to determine what might have caused it.

javax.naming.NameNotFoundException: Name name not found in context "local": A possible reason for this is exception is that the enterprise bean is not local, (not running in the same Java virtual machine [JVM] or Application Server), to the client JSP, servlet, Java application, or other enterprise bean, yet the call is to one of the enterprise bean's "local" interface methods. If access worked in a development environment but not when deployed to WebSphere Application

Server, for example, it could be that the enterprise bean and its client were in the same JVM in development, but after deployment they are in separate processes.

To resolve this problem, contact the developer of the enterprise bean and determine whether the client call is to a method in the enterprise bean's local interface. If so, have the client code changed to call a remote interface method, or promote the local method into the remote interface.

BeanNotReentrantException is thrown: This problem can be caused by client code (typically a servlet or JSP) attempting to call the same stateful SessionBean from two different client threads. This situation often arises when the an application stores the reference to the stateful session bean in a static variable, uses a global (static) JSP variable to refer to the stateful SessionBean reference, or stores the stateful SessionBean reference in the HTTP session object and then has the client browser issue a new request to the servlet or JSP before the previous request has completed.

To resolve this problem, ask the developer of the client code to review their code for these conditions.

CSITransactionRolledbackException / TransactionRolledbackException is thrown: These are high-level exceptions thrown by an enterprise bean's container, and indicate that an enterprise bean call could not be successfully completed. When this exception is thrown, browse the JVM logs to determine the underlying cause.

Some possible causes are:
- The enterprise bean may be throwing an exception that was not declared as part of its method signature. The container is required to roll back the transaction in this case. Common causes of this situation are where the enterprise bean or code that it calls throws a NullPointerException, ArrayIndexOutOfBoundsException, or other Java "runtime" exception, or where a BMP bean encounters a JDBC error. The resolution is to investigate the enterprise bean code and resolve the underlying exception, or to add the exception to the problem method's signature.
- A transaction may have attempted to do additional work after being placed in a "Marked Rollback", "RollingBack", or "RolledBack" state. Transactions cannot continue to do work after they have been set to one of these states. Often this occurs because the transaction has timed out which, in turn, often occurs because of a database deadlock. The resolution is to work with the application's database managements tools or administrator to determine whether database transactions called by the enterprise bean are timing out.
- A transaction may fail on commit due to "dangling work". This could be due to "local" transactions. The local transaction encountered some "dangling work" during commit. The default "action" for local transactions when they encounter an "Unresolved Action" is to "rollback". This can be adjusted to "commit" in the Application Assembly Tool. In the AAT, open the enterprise bean `.jar` file (or the EAR file containing the enterprise bean) and select the "Session Beans" or "Entity Beans" object in the component tree on the left. The "Unresolved Action" property is on the "IBM Extensions" tab of the container properties.

Attempt to start EJB module fails with "javax.naming.NameNotFoundException dataSourceName_CMP"exception: The possible causes of this problem are:
- When the DataSource resource was configured, Container Managed Persistence was not selected.

- To confirm that this is the problem, in the administrative console, browse the properties of the data source given in the NameNotFoundException. On the Configuration panel, look for a checkbox labeled **Container Managed Persistence**.
  - To correct this problem, select checkbox for **Container Managed Persistence**.
- If Container Managed Persistence is selected, it is possible that the CMP DataSource could not be bound into the namespace.
  - Look for additional naming warnings or errors in the status bar, and in the hosting application server's JVM logs. Check any further naming-exception problems that you find by looking at the topic Cannot access an object hosted by WebSphere Application Server (enterprise bean, connection pool, etc) from a servlet, jsp, standalone program , or other client.

Transaction [tran ID] has timed out after 120 seconds accessing EJB: This error can happen when a client executes a transaction on a CMP or BMP enterprise bean.

- The default timeout value for enterprise bean transactions is 120 seconds. After this time, the transaction times out and the connection is closed.
- If the transaction legitimately takes longer than the specified timeout period, go to **Manage Application Servers -> *server_name***, select the **Transaction Service properties** page, and look at the property **Total transaction lifetime timeout**. Increase this value if necessary and save the configuration.

Symptom:CNTR0001W: A Stateful SessionBean could not be passivated: This error can occur when a Connection Object being used in the bean has not been closed or nulled out.

To confirm that this is the problem, look for an exception stack in the JVM log for the EJB Container which hosts the enterprise bean, which looks similar to:

```
[time EDT] <ThreadID> StatefulPassi W CNTR0001W: A Stateful SessionBean could not
be passivated: StatefulBeanO(BeanId(XXX#YYY.jar#ZZZZ, <ThreadID>),
state = PASSIVATING) com.ibm.ejs.container.passivator.StatefulPassivator@
<ThreadID> java.io.NotSerializableException:
com.ibm.ws.rsadapter.jdbc.WSJdbcConnection
at java.io.ObjectOutputStream.outputObject((Compiled Code))
at java.io.ObjectOutputStream.writeObject(ObjectOutputStream.java(Compiled Code))
at java.io.ObjectOutputStream.outputClassFields((Compiled Code))
at java.io.ObjectOutputStream.defaultWriteObject((Compiled Code))
at java.io.ObjectOutputStream.outputObject((Compiled Code))
at java.io.ObjectOutputStream.writeObject(ObjectOutputStream.java(Compiled Code))
at com.ibm.ejs.container.passivator.StatefulPassivator.passivate((Compiled Code))
at com.ibm.ejs.container.StatefulBeanO.passivate((Compiled Code)
at com.ibm.ejs.container.activator.StatefulASActivationStrategy.
atUnitOfWorkEnd((Compiled Code))
at com.ibm.ejs.container.activator.Activator.unitOfWorkEnd((Compiled Code))
at com.ibm.ejs.container.ContainerAS.afterCompletion((Compiled Code)
```

where XXX,YYY,ZZZ is the Bean's name, and <ThreadID> is the thread ID for that run.

To correct this problem, the application must close all connections and set the reference to null for all connections. Typically this is done in the `ejbPassivate()` method of the bean. See the enterprise bean specification mandating this requirement, specifically section 7.4 in the EJB specification version 2.0. Also, note that the bean must be coded to re-acquire these connections when the bean is re-activated. Otherwise, there will be NullPointerExceptions when the application tries to re-use the connections.

Cannot look up an object hosted by WebSphere Application Server from a servlet, JSP file, or other client

To resolve problems encountered when a servlet, JSP file, standalone application or other client attempts to access an enterprise bean, ConnectionPool, or other named object hosted by WebSphere Application Server, you must first verify that the target server can be accessed from the client:

- From a command prompt on the client's server, enter "ping <server_name>" and verify connectivity.
- Use the WebSphere Application Server administrative console to verify that the target resource's application server and, if applicable, EJB module or Web module, is started.

Continue only if there is no problem with connectivity and the target resource appears to be running.

What kind of error are you seeing?

- NameNotFoundException from JNDI lookup operation
- CannotInstantiateObjectException from JNDI lookup operation
- Message NMSV0610I appears in the server's log file, indicating that some Naming exception has occurred
- OperationNotSupportedException from JNDI Context operation.
- "WSVR0046E: Failed to bind" error, with Original exception: "org.omg.CosNaming.NamingContextPackage.AlreadyBound".
- ConfigurationException from "new InitialContext" operation or from a JNDI Context operation with a URL name.
- ServiceUnavailableException from "new InitialContext" operation.
- CommunicationException thrown from a "new InitialContext" operation.
- NMSV0605E: A Reference object looked up from the context...

If you do not see a problem that resembles yours, or if the information provided does not solve your problem, contact IBM support for further assistance.

NameNotFoundException from JNDI lookup operation: If you encounter this exception in trying to access an enterprise bean, data source, messaging resource, or other resource:

- Browse the properties of the target object in the administrative console, and verify that the jndi name it specifies matches the JNDI name the client is using.
- If you are looking up an object that resides on a server different from the one from which he initial context was obtained, you must use the fully qualified name.
  - If access is from another server object such as a servlet accessing an enterprise bean and you are using the default context, not specifying the fully qualified JNDI name, you may get this error if the object is being hosted on a different server.
  - If access is from a standalone client, it may be that the object you are attempting access is on a server different from the server from which you obtained the initial context.

To correct this problem, use the fully-qualified JNDIname:

- If the object is in a single server:
  cell/nodes/<nodeName>/servers/<serverName>/<jndiName>. Objects are not supported in this release.

- If the object is on a server cluster: `cell/clusters/<clusterName>/<jndiName>`.

CannotInstantiateObjectException from JNDI lookup operation: If you encounter this exception in trying to access an enterprise bean, data source, messaging resource, or other resource, possible causes include:

- A serialized Java object is being looked up, but the necessary classes required to deserialize it are not in the runtime environment.
- A Reference object is being looked up, and the associated factory used to process it as part of the lookup processing is failing.

To determine the precise cause of the problem:

- Look in the JVM logs of the server hosting the target resource. Look for exceptions immediately preceding the CannotInstantiateObjectException. If it is a java.lang.NoClassDefFoundError or java.lang.ClassNotFoundException, make sure the class referenced in the error message can be located by the class loader.
- Print out the stack trace for the root cause and look for the factory class. It will be called by `javax.naming.NamingManager.getObjectInstance()`. The reason for the failure will depend on the factory implementation, and may require you to contact the developer of the factory class.

Message NMSV0610I appears in the server's log file, indicating that some Naming exception has occurred: This error is informational only and is provided in case the exception is related to an actual problem. Most of the time, it is not. If it is, the log file should contain adjacent entries to provide context.

- If no problems are being experienced, ignore this message. Also ignore the message if the problem you are experiencing does not appear to be related to the exception being reported and if there are no other adjacent error messages in the log.
- If a problem is being experienced, look in the log for underlying error messages.
- The information provided in message NMSV0610I can provide valuable debug data for other adjacent error messages posted in response to the Naming exception that occurred.

OperationNotSupportedException from JNDI Context operation: This error has two possible causes:

- An update operation, such as a bind, is being performed with a name that starts with `"java:comp/env"`. This context and its subcontexts are read-only contexts.
- A Context bind or rebind operation of a non-CORBA object is being performed on a remote name space that does not belong to WebSphere Application Server. Only CORBA objects can be bound to these CosNaming name spaces.

To determine which of these errors is causing the problem, check the full exception message.

WSVR0046E: Failed to bind, ejb/jndiName: ejb/jndiName. Original exception : org.omg.CosNaming.NamingContextPackage.AlreadyBound: This error occurs two enterprise bean server applications were installed on the same server such that a binding name conflict occurred. That is, a jndiName value is the same in the two applications' deployment descriptors. The error will surface during server startup when the second application using that jndiName value is started.

To verify that this is the problem, examine the deployment descriptors for all enterprise bean server applications running in the server in search for a jndiName that is specified in more than one enterprise bean application.

To correct the problem, change any duplicate jndiName values to ensure that each enterprise bean in the server process is bound with a different name.

ConfigurationException from "new InitialContext" operation or from a JNDI Context operation with a URL name: If you are attempting to obtain an initial JNDI context, a configuration exception can occur because an invalid JNDI property value was passed to the InitialContext constructor. This includes JNDI properties set in the System properties or in some jndi.properties file visible to the class loader in effect. A malformed provider URL is the most likely property to be incorrect. If the JNDI client is being run as a thin client such that the CLASSPATH is set to include all of the individual jar files required, make sure the .jar file containing the properties file com/ibm/websphere/naming/jndiprovider.properties is in the CLASSPATH.

If the exception is occurring from a JNDI Context call with a name in the form of a URL, the current JNDI configuration may not be set up properly so that the required factory class name cannot be determined, or the factory may not be visible to the class loader currently in effect. If the name is a Java: URL, the JNDI client must be running in a J2EE client or server environment. That is, the client must be running in a container.

Check the exception message to verify the cause.

If the exception is being thrown from the InitialContext constructor, correct the property setting or the CLASSPATH.

If the exception is being thrown from a JNDI Context method, make sure the property java.naming.factory.url.pkgs includes the package name for the factory required for the URL scheme in the name. URL names with the Java scheme can only be used while running in a container.

ServiceUnavailableException from "new InitialContext" operation: This exception indicates that some unexpected problem occurred while attempting to contact the name server to obtain an initial context. The ServiceUnavailableException, like all NamingException objects, can be queried for a root cause. Check the root cause for more information. It is possible that some of the problems described for CommunicationExceptions may also result in a ServiceUnavailableException.

Since this exception is triggered by an unexpected error, there is no probable cause to confirm. If the root cause exception does not indicate what the probable cause is, investigate the possible causes listed for CommunicationExceptions.

CommunicationException thrown from a "new InitialContext" operation: The name server identified by the provider URL cannot be contacted to obtain the initial JNDI context. There are many possible causes for this problem, including:
- The host name or port in the provider URL is incorrect.
- The host name cannot be resolved into an IP address by the domain name server, or the IP address does not match the IP address which the server is actually running under.
- A firewall on the client or server is preventing the port specified in the provider URL from being used.

To correct this problem:

- Make sure the provider URL and the network configurations on the client and server machines are correct.
- Make sure the host name can be resolved into an IP address which can be reached by the client machine. You can do this using the ping command.
- If you are running a firewall, make sure that use of the port specified in the provider URL will be allowed.

Errors or access problems after enabling security

What kind of error are you seeing?

- I cannot access part or all of administrative console or use wsadmin after enabling security
- I cannot access a Web page after enabling security
- The client cannot access an enterprise bean after enabling security
- The client never gets prompted when accessing a secured enterprise bean
- I cannot stop an application server, node manager, or node after enabling security
- Error Message: SECJ0314E: Current Java 2 Security policy reported a potential violation
- "MSGS0508E: The JMS Server security service was unable to authenticate userid:" error displayed in SystemOut.log when starting an application server
- "SECJ0237E: One or more vital LTPAServerObject configuration attributes are null or not available" after enabling security and starting application server.
- AccessControlException is reported in SystemOut.log.
- After enabling single sign-on, I cannot log on to the administrative console.

For general tips on diagnosing and resolving security-related problems, see the topic Troubleshooting the security component.

If you do not see a problem that resembles yours, or if the information provided does not solve your problem, contact IBM support for further assistance.

Cannot access part or all of admin console or use wsadmin after enabling security:

- If you cannot access the administrative console, or view or update certain objects, look in the SystemOut log of the the application server which hosts the administrative console page for a related error message.
- You may not have authorized your ID for administrative tasks. This is indicated by errors such as:
  - **[8/2/02 10:36:49:722 CDT] 4365c0d9 RoleBasedAuth A SECJ0305A: Role based authorization check failed for security name MyServer/myUserId, accessId MyServer/S-1-5-21-882015564-4266526380-2569651501-1005 while invoking method getProcessType on resource Server and module Server.**
  - Exception message: **"ADMN0022E: Access denied for the getProcessType operation on Server MBean"**
  - **When running the command: wsadmin -username j2ee -password j2ee: WASX7246E: Cannot establish "SOAP" connection to host "BIRKT20" because of an authentication failure. Please ensure that user and password are correct on the command line or in a properties file.**
- To grant an ID administrative authority:
  - From the Administrative Console, select **Security Center -> Manage Console Users** and validate that the ID is a member. If it is not, add the ID with at least monitor access privileges, for read-only access.

Cannot access a web page after enabling security: When secured resources cannot be accessed, these are some possible causes:

- Authentication errors - WebSphere Application Server security cannot identify the ID of the person or process. Symptoms of authentication errors include:
  - Netscape browser:
    - "Authorization failed. Retry?" message displayed after an attempt to login.
    - Allows any number of attempts to retry login and displays "Error 401" message when "cancel" is pressed to stop retry.
    - Typical browser message: "Error 401: Basic realm='Default Realm'".
  - Internet Explorer browser:
    - Login prompt displayed again after an attempt to login.
    - Allows 3 attempts to retry login.
    - Displays "Error 401" message after 3 unsuccessful retries.
- Authorization errors - security has identified the requesting person or process as not authorized to access the secured resource. Symptoms of authorization errors include:
  - Netscape browser: "Error 403: AuthorizationFailed" message is displayed.
  - Internet Explorer:
    - "You are not authorized to view this page" message is displayed.
    - "HTTP 403 Forbidden" error is also displayed.
- SSL errors - WebSphere Application Server security uses Secure Socket Layer (SSL) technology internally to secure and encrypt its own communication, and misconfiguration of the internal SSL settings can cause problems. Also you might have enabled SSL encryption for your own Web application or enterprise bean client traffic which, if misconfigured, can cause problems regardless of whether WebSphere Application Server security is enabled. SSL related problems are often indicated by error messages which contain a statement such as:
  - ERROR: Could not get the initial context or unable to look up the starting context. Exiting. followed by:
  - javax.net.ssl.SSLHandshakeException

Authentication error accessing a web page: Possible causes for authentication errors are:

Username or Passwords invalid
> Check the username and password and make sure they are correct.

Security Configuration error : User registry type is not set correctly.
> Check the User Registry property in the global security settings in the administrative console. Ensure that it is the intended User Registry.

Internal program error
> If client application is a Java standalone program, it may not be gathering or sending the credential information correctly.

If the User Registry configuration, and user ID and password, appear to be correct, use the WebSphere Application Server trace to determine the cause of the problem. To enable security trace use the com.ibm.ws.security.*=all=enabled trace specification.

Authorization error accessing a Web page: If a user who should have access to a resource does not, the problem is probably due to a missing configuration step. Review the steps for securing and granting access to resources.

Specifically:

- Check the required roles for the Web Resource being accessed.
- Check the authorization table and make sure that the user or the groups that the user belongs to is assigned one of the required roles.
- Required roles for the Web Resource can be viewed in the deployment descriptor of the Web Resource.
- The authorization table for the application that contains the Web Resource can also be viewed using administrative console.
- Try using a user that is granted the required roles to see if that user can access the problem resources.
- If the problem user is required to be granted one or more of the required roles, use the administrative console to assign that user to required roles, and then stop and restart the application.

If user is granted the required roles and still fails to access the secured resources, enable security trace using com.ibm.ws.security.*=all=enabled as the trace specification and retry to collect the trace information for further resolution.

Cannot access an enterprise bean after enabling security: If client access to an enterprise bean fails after security is enabled:

- Review the steps for securing and granting access to resources.
- Browse the server's JVM logs for errors relating to enterprise bean access and security. Look up any errors in the message table.
- Errors similar to **Authorization failed for /UNAUTHENTICATED while invoking** *resource* **securityName:/UNAUTHENTICATED;accessId:UNAUTHENTICATED not granted any of the required roles** *roles* indicate that:
  - an unprotected servlet or JSP accessed a protected enterprise bean. When unprotected servlet is accessed, the user is not prompted to login and hence the servlet runs as UNAUTHENTICATED. When it makes a call to an enterprise bean that is protected it will fail.

    To resolve this problem, secure the servlet that is accessing the secured enterprise bean. Make sure the servlet's `runAs` property is set to an ID that can access the enterprise bean.
  - An unauthenticated Java client program is accessing an enterprise bean resource that is protected. This can happen if the file read by the `sas.client.props` properties file used by the client program does not have the `securityEnabled` flag set to `true`.

    To resolve this problem, make sure that the `sas.client.props` file on the client side has its securityEnabled flag set to true.
- Errors similar to **Authorization failed for** *valid_user* **while invoking** *resource* **securityName:/username;accessId:xxxxxx not granted any of the required roles** *roles* indicate that a client attempted to access a secured enterprise bean resource, and the supplied user ID is not assigned the required roles for that enterprise bean.
  - Check the required roles for the enterprise bean resource being accessed. Required roles for the enterprise bean resource can be viewed in the deployment descriptor of the Web resource.
  - Check the authorization table and make sure that the user or the group that the user belongs to is assigned one of the required roles. The authorization table for the application that contains the enterprise bean resource can also be viewed using administrative console.

org.omg.CORBA.NO_PERMISSION exceptions returned when programmatically logging on in order to access a secured EJB indicate an authentication exception has occurred on the server. Typically the CORBA exception is triggered by an underlying `com.ibm.WebSphereSecurity.AuthenticationFailedException`. To determine the actual cause of the authentication exception, the full trace stack must be examined.

- Begin by viewing the text in the exception, following "WSSecurityContext.acceptSecContext(), reason:". Typically, it describes the failure without further analysis.
- If this does not describe enough of the problem, look up the CORBA minor code. Those codes are listed in the Troubleshooting the security components reference in this document.

For example, the following exception:

org.omg.CORBA.NO_PERMISSION: Caught WSSecurityContextException in WSSecurityContext.acceptSecContext(), reason: Major Code[0] Minor Code[0] Message[Exception caught invoking authenticateBasicAuthData from SecurityServer for user jdoe.
Reason: com.ibm.WebSphereSecurity.AuthenticationFailedException] minor code: 49424300 completed: No   at com.ibm.ISecurityLocalObjectBaseL13Impl. PrincipalAuthFailReason.map_auth_fail_to_minor_code(PrincipalAuthFailReason.java:83)

indicates a CORBA minor code of 49424300. The explanation of this error in the CORBA minor code table reads:

`authentication failed error.`

In other words, in this case the user ID or password supplied by the client program is probably invalid.

CORBA INITIALIZE exception with **JSAS1477W: SECURITY CLIENT/SERVER CONFIG MISMATCH** error embedded, received by client program from server.

This error indicates that the server's security configuration differs from the client in some fundamental way. The full exception message will list the specific mismatches. For example, the following exception lists three:

```
Exception received: org.omg.CORBA.INITIALIZE:

JSAS1477W: SECURITY CLIENT/SERVER CONFIG MISMATCH: The client security
configuration (sas.client.props or outbound settings in GUI) does not
support the server security configuration for the following reasons:
ERROR 1: JSAS0607E: The client requires SSL Confidentiality but the server does not support it.
ERROR 2: JSAS0610E: The server requires SSL Integrity but the client does not support it.
ERROR 3: JSAS0612E: The client requires client (e.g., userid/password or token),
but the server does not support it.
minor code: 0 completed: No at
com.ibm.ISecurityLocalObjectBaseL13Impl.SecurityConnectionInterceptor.
getConnectionKey(SecurityConnectionInterceptor.java:1770)
```

In general, resolving the problem requires a change to the security configuration of either the client or the server. In order to determine which configuration setting is involved, look at the text following the "JSAS" error message. For more detailed explanations and instructions, look up the error message in the message reference, found in the "Quick Reference" view of this InfoCenter.

In these particular cases:

- In the case of ERROR 1, the client is requiring SSL Confidentiality but the server does not support SSL Confidentiality. This can be resolved in two ways. The server either supports it or the client no longer requires it.
- In the case of ERROR 2, the server requires SSL Integrity but the client does not support SSL Integrity. Again, there are two ways this problem can be solved. Get the server to not require integrity or get the client to support integrity.
- Finally, in the case of ERROR 3, the client requires client authentication via userid and password, but the server does not support this type of client authentication. Again, either the client or the server needs to change the configuration. To change the client configuration, modify the `SAS.CLIENT.PROPS` file for a pure client or change the server's outbound configuration in the Security GUI. To change the target server's configuration, modify the inbound configuration in the Security GUI.

Similarly, an exception like `org.omg.CORBA.INITIALIZE: JSAS0477W: SECURITY CLIENT/SERVER CONFIG MISMATCH:` appearing on the server trying to service a client request indicates a security configuration mismatch between client and server. The steps for resolving the problem is the same as for JSAS1477W exceptions described above.

Client program never gets prompted when accessing secured enterprise bean: Even though it appears security is enabled and an enterprise bean is secured, it may happen that the client executes the remote method without getting prompted.
- If the remote method is protected, you should get an authorization failure. Otherwise,
- Execute the method as an unauthenticated user.

Possible reasons for this include:
- The server you are communicating with may not have security enabled. Check with the WebSphere Application Server administrator to ensure that the server security is enabled. This is done in the global security settings from within the Security section of the administrative console.
- The client does not have security enabled in the `sas.client.props` file. Edit the `sas.client.props` to ensure the property `com.ibm.CORBA.securityEnabled=true`.
- The client does not have a ConfigURL specified. Ensure that the property `com.ibm.CORBA.ConfigURL` is specified on the command line of the Java client, using the -D parameter.
- The specified ConfigURL has an invalid URL syntax or the `sas.client.props` pointed to by it cannot be found. Ensure that the property `<strong>com.ibm.CORBA.ConfigURL</strong>` is valid, for example, similar `tofile:/C:/WebSphere/AppServer/properties/sas.client.props` on Windows systems. Check the Java documentation for a description of URL formatting rules. Also, validate that the file exists at the specified path.
- The client configuration does not support message layer client authentication (userid and password). Ensure that the `sas.client.props` has one of the following properties set to true:
  - `<strong>com.ibm.CSI.performClientAuthenticationSupported=true </strong>`
  - `<strong>com.ibm.CSI.performClientAuthenticationRequired=true. </strong>`
- The server configuration does not support message layer client authentication (userid and password). Check with the WebSphere Application Server administrator to ensure that Userid and Password authentication is specified for

the Inbound configuration of the server within the System Administration section of the administrative console administration tool.

Cannot stop an application server, node manager, or node after enabling security: If you are using command line utilities to stop WAS processes, you need to apply additional parameters after enabling security, in order to provide authentication and authorization information.

Use the command: **./stopServer.sh -help** to display the parameters that should be used.

You should use the following command options after enabling security:
- ./stopServer.sh hostname -username *name* -password *password*
- ./stopNode.sh -username *name* -password *password*
- ./stopManager.sh -username *name* -password *password*

Error Message: SECJ0314E: Current Java 2 Security policy reported a potential violation on server: If you find errors on your server similar to:

```
Error Message: SECJ0314E: Current Java 2 Security policy reported a potential
violation of Java 2 Security Permission. Please refer to Problem Determination Guide
for further information.{0}Permission\:{1}Code\:{2}{3}Stack Trace\:{4}Code Base Location\:{5}
```

then The Java Security Manager `checkPermission()` method has reported a `SecurityException`.

The reported exception may be critical to the secure system. Turn on security trace to determine the potential code that may have violated the security policy. Once the violating code is determined, you should verify if the attempted operation is permitted with respect to Java 2 Security, by examining all applicable Java 2 security policy files and the application code itself.

A more detailed report is enabled by either configuring RAS trace into debug mode, or specifying a Java property.
- Please check the trace enabling section for instructions on how to configure RAS trace into debug mode, or
- Specify the following property in the **Application Servers >** *server name* **> ProcessDefinition > Java virtual machine** panel from the administrative console in the **Generic JVM arguments** panel:
  - add the runtime flag `<strong>java.security.debug</strong>`
  - Valid values:

    access to print all debug information including: required permission, code, stack, and code base location.

    stack to print debug information including: required permission, code, and stack.

    failure

    to print debug information including: required permission and code.

For a review of Java security policies and what they mean , see the Java 2 Security documentation at `http://java.sun.com/j2se/1.3/docs/guide/security/index.html`
.

Note: If the application is running with Java Mail, this message may be benign. You can update the `<installed Enterprise Application root>/META-INF/was.policy` file to grant the following permissions to the application:

- `permission java.io.FilePermission "${user.home}${/}.mailcap", "read";`
- `permission java.io.FilePermission "${user.home}${/}.mime.types", "read";`
- `permission java.io.FilePermission "${java.home}${/}lib${/}mailcap", "read";`
- `permission java.io.FilePermission "${java.home}${/}lib${/}mime.types", "read";`

"**MSGS0508E: The JMS Server security service was unable to authenticate userid:**" error displayed in SystemOut.log when starting an application server: This error may be a result of installing the JMS (messaging api) sample and then enabling security. The JMS sample is not designed to work with WebSphere Application Server security. If WebSphere Application Server was installed with samples and no additional code was installed which uses messaging, this message may be ignored.

You can verify the installation of the message-driven bean sample by launching the installation program, selecting **Custom**, and browsing the components which are already installed in the **Select the features you like to install** panel. The JMS sample is shown as **Message-Driven Bean Sample**, under **Embedded Messaging**.

You can also verify this by using the administrative console to open the properties of the application server which contains the samples, selecting "MDBSamples" and clicking "uninstall".

If the problem persists, review the section Troubleshooting JMS.

SECJ0237E: One or more vital LTPAServerObject configuration attributes are null or not available: The most likely cause of this error is that LTPA is selected as authentication mechanism but the LTPA keys have not been generated. The LTPA keys are used for encrypting the LTPA token.

To resolve this problem:
1. Select **System Administration -> Console users -> LTPA**
2. Enter a password, which can be anything.
3. Enter the same password in "Confirm Password".
4. Click **Apply**.
5. Click **Generate Keys**.
6. Click on **Save**.

AccessControlException is reported in SystemOut.log: If you see an exception similar to the following (error message and number may vary):

```
E SRVE0020E: [Servlet Error]-[validator]: Failed to load servlet:
java.security.AccessControlException:
access denied (java.io.FilePermission
C:\WebSphere\AppServer\installedApps\maeda\adminconsole.ear\adminconsole.war
\WEB-INF\validation.xml read)
```

the problem is related to the **Java 2 Security** feature of WebSphere Application Server, the api-level security framework that is implemented in WebSphere Application Server Version 5. For an explanation of Java 2 security, how and why

to enable or disable it, and how it relates to policy files, and how to edit policy files, see the topic Java 2 Security in this InfoCenter. As this topic explains, Java 2 security is not only used by this product, but may also be implemented by business application developers, and administrators may need to involve developers if this exception is thrown when a client tries to access a resource hosted by WebSphere Application Server.

Some possible causes of these errors are:

- Syntax errors in a policy file.
- Syntax errors in permission specifications in the ra.xml file bundled in a .rar file. This case applies to resource adapters which support "connector" access to CICS or other resources.
- An application is missing the specified permission in a policy file, or in permission specifications in an ra.xml file bundled in a .rar file
- The classpath was not set correctly. If the classpath is not set correctly in resource.xml file for SPI, permissions cannot be created correctly.
- A library called by an application, or the application itself, is missing a doPrivileged block to allow access to a resource.
- Permission is specified in the wrong policy file.

To resolve these problems:

- Check all of the related policy files to ensure that the permission shown in the exception, for example java.io.FilePermission, is specified.
- Look for a related ParserException in SystemOut.log which reports the details of the syntax error. For example: **SECJ0189E: Caught ParserException** while creating template for Application Policy C:\WAS\config\cells\xxx\nodes\xxx\app.policy. The exception is com.ibm.ws.security.util.ParserException: line 18: expected ';', found 'grant'
- Look for a message similar to: **SECJ0325W: The permission** *permission* **specified in the policy file is unresolved.**
- Check the call stack to determine which method does not have the permission. Identify what this method's classpath is. If it is hard to identify the method, enable **Java2 security Report**.
  - Configuring RAS trace by specifying `com.ibm.ws.security.core.*=all=enabled`, or specifying a Java `property.java.security.debug` . Valid values for property java.security.debug are:

    access to print all debug information including: required permission, code, stack, and code base location.

    stack to print debug information including: required permission, code, and stack.

    failure
    to print debug information including: required permission and code.
  - The report shows:

    Permission
    the missing permission.

    Code which method has the problem.

    Stack Trace
    where the access violation occurred.

CodeBaseLocation
         the detail of each stack frame.

Usually, Permission and Code should be enough to identify the problem. The following is an example of a report:

```
Permission:
      C:\WebSphere\AppServer\logs\server1\SystemOut_02.08.20_11.19.53.log :
access denied (java.io.FilePermission
C:\WebSphere\AppServer\logs\server1\SystemOut_02.08.20_11.19.53.log delete)

Code:
      com.ibm.ejs.ras.RasTestHelper$7  in
{file:/C:/WebSphere/AppServer/installedApps/maeda/JrasFVTApp.ear/RasLib.jar}

Stack Trace:

java.security.AccessControlException: access denied (java.io.FilePermission
C:\WebSphere\AppServer\logs\server1\SystemOut_02.08.20_11.19.53.log delete)
at java.security.AccessControlContext.checkPermission
(AccessControlContext.java(Compiled Code))
at java.security.AccessController.checkPermission(AccessController.java(Compiled Code))
at java.lang.SecurityManager.checkPermission(SecurityManager.java(Compiled Code)).
                                                  .
Code Base Location:

com.ibm.ws.security.core.SecurityManager :
  file:/C:/WebSphere/AppServer/lib/securityimpl.jar
  ClassLoader: com.ibm.ws.bootstrap.ExtClassLoader
  Permissions granted to CodeSource
  (file:/C:/WebSphere/AppServer/lib/securityimpl.jar
<no certificates>
  {
    (java.util.PropertyPermission java.vendor read);
    (java.util.PropertyPermission java.specification.version read);
    (java.util.PropertyPermission line.separator read);
    (java.util.PropertyPermission java.class.version read);
    (java.util.PropertyPermission java.specification.name read);
    (java.util.PropertyPermission java.vendor.url read);
    (java.util.PropertyPermission java.vm.version read);
    (java.util.PropertyPermission os.name read);
    (java.util.PropertyPermission os.arch read);
  }
  ( This list continues.)
```

- If the method is SPI, check the **resources.xml** file to ensure that the classpath is correct.
- In order to confirm that all of the policy files are loaded correctly, or what permission each classpath is granted, **enable the trace with `com.ibm.ws.security.policy.*=all=enabled`** . All of the loaded permission will be listed in **trace.log**. Search for app.policy,was.policy and ra.xml. In order to check the permission list for a classpath, search for Effective Policy for <classpath>.
- If there is any syntax error in the policy file or ra.xml file, correct it with policytool. Please avoid editing the policy manually, since it can cause syntax errors.
- If a permission is listed as **Unresolved** it does not take effect. Please make sure that the specified permission name is correct.
- If the classpath specified in resource.xml file is not correct, correct it.
- If a required permission does not exist in policy files or the ra.xml file, examine the application code to see if this permission needs to be added. If so, add it to proper policy file or ra.xml file.

- If the permission should not be granted outside of the specific method that is accessing this resource, the code needs to be modified to use a doPrivileged block.
- If this permission does exist in a policy file or `ra.xml` file and they were loaded correctly, but the classpath still does not have the permission in its list, the location of the permission may not be correct. Please read Java 2 Security in this InfoCenter carefully to determine in which policy file or `ra.xml` file that permission should be specified.

Note: If the application is running with Java Mail, this message may be benign. You can update the `<installed Enterprise Application root>/META-INF/was.policy` file to grant the following permissions to the application:
- `permission java.io.FilePermission "${user.home}${/}.mailcap", "read";`
- `permission java.io.FilePermission "${user.home}${/}.mime.types", "read";`
- `permission java.io.FilePermission "${java.home}${/}lib${/}mailcap", "read";`
- `permission java.io.FilePermission "${java.home}${/}lib${/}mime.types", "read";`

After enabling single sign-on I cannot log on to the administrative console: This problem occurs when single sign-on (SSO) is enabled, and you attempt to access the administrative console using the short name of the server, for example `http://myserver:9090/admin`. The server will accept your userID and password, but returns you to the sign-on page instead of the administrative console.

To correct this problem, use the fully-qualified hostname of the server, for example `http://myserver.mynetwork.mycompany.com:9090/admin`.

Errors after enabling SSL, or SSL-related error messages

If you are unable to access resources using a Secure-socket layer (SSL) type URL (beginning with "https:"), or encounter error messages which indicate SSL problems, ensure that your HTTP server has been configured correctly for SSL by browsing the welcome page of the HTTP server using SSL by entering the URL **https://** *hostname*.

If the page works with HTTP, but not HTTPS, the problem is in the HTTP server.
- Refer to the documentation for your HTTP server for instructions on correctly enabling SSL. If you are using IHS or Apache, go to: http://www.ibm.com/software/webservers/httpservers/library.html. Select the link *Frequently Asked Questions*, and the topic *SSL*.
- If you are using the **IKeyman** (IBM Key Management) tool to create certificates and keys, remember to "stash" the password to a file when creating the KDB file with the IBM Key Management Tool.
    1. Go to the directory where the KDB file was created, and check to see if there is a .sth file. If not,
    2. Open the KDB file with the IBM Key Management Tool, select **Key Database File > Stash Password**.
    3. It will display "The password has been encrypted and saved in the file".

If the HTTP server handles SSL-encrypted requests successfully, or is not involved (for example, traffic flows from a java client application directly to an enterprise bean hosted by the WebSphere Application Server, or the problem appears only after enabling WebSphere Application Server security), what kind of error are you seeing?

- javax.net.ssl.SSLHandshakeException - The client and server could not negotiate the desired level of security. Reason: handshake failure
- javax.net.ssl.SSLHandshakeException - The client and server could not negotiate the desired level of security. Reason: unknown certificate
- javax.net.ssl.SSLHandshakeException - The client and server could not negotiate the desired level of security. Reason: bad certificate
- org.omg.CORBA.INTERNAL: EntryNotFoundException or NTRegistryImp E SECJ0070E: No privilege id configured for: error when programmatically creating a credential.

For general tips on diagnosing and resolving security-related problems, see the topic Troubleshooting the security component.

If you do not see a problem that resembles yours, or if the information provided does not solve your problem, contact IBM support for further assistance.

javax.net.ssl.SSLHandshakeException - The client and server could not negotiate the desired level of security. Reason: handshake failure: If you see a java exception stack similar to:

[Root exception is org.omg.CORBA.TRANSIENT: CAUGHT_EXCEPTION_WHILE_CONFIGURING_SSL_CLIENT_SOCKET: JSSL0080E: javax.net.ssl.SSLHandshakeException - The client and server could not negotiate the desired level of security.
Reason: handshake failure:host=MYSERVER,port=1079 minor code: 4942F303 completed: No]
at com.ibm.CORBA.transport.TransportConnectionBase.connect (TransportConnectionBase.java:NNN)

some possible causes are:
- Not having common ciphers between the client and server.
- Not specifying the correct protocol.

To correct these problems:
- Review the SSL settings by browsing the **WebSphere Administrative Console Security Settings -> SSL Configuration Repertoires -> DefaultSSLSettings** (or other named SSL settings), then selecting the **Secure Socket Layer (SSL)** option from the **Additional Properties** menu. You can also browse the file manually by viewing: *install_dir*/properties/sas.client.props.
- Check the property specified by **com.ibm.ssl.protocol** to determine which protocol is specified.
- Check the cipher types specified by **com.ibm.ssl.enabledCipherSuites**. You may want to add more cipher types to the list. To see which cypher suites are currently enabled, go to the properties page of the SSL settings as described above, and look for the **Cipher Suites** property. To see the list of all possible cipher suites, go to the properties page of the SSL settings as described above, then view the online help for that page. From the help page, click **Configure additional SSL settings**.
- Correct the protocol or cipher problem by using a different client or server protocol and/or cipher selection. Typical protocols are SSL or SSLv3.
- Make the cipher selection 40-bit instead of 128-bit. For CSIv2, set both of these properties to false:
  - **com.ibm.CSI.performMessageConfidentialityRequired=false**

- **com.ibm.CSI.performMessageConfidentialitySupported=false**

in the sas.client.props file, or set the **security level=medium** in the Administrative Console settings.

javax.net.ssl.SSLHandshakeException: unknown certificate: If you see a java exception stack similar to: **ERROR: Could not get the initial context or unable to look up the starting context. Exiting. Exception received: javax.naming.ServiceUnavailableException: A communication failure occurred while attempting to obtain an initial context using the provider url: ″corbaloc:iiop:localhost:2809″. Make sure that the host and port information is correct and that the server identified by the provider url is a running name server. If no port number is specified, the default port number 2809 is used. Other possible causes include the network environment or workstation network configuration. [Root exception is org.omg.CORBA.TRANSIENT: CAUGHT_EXCEPTION_WHILE_CONFIGURING_SSL_CLIENT_SOCKET: JSSL0080E: javax.net.ssl.SSLHandshakeException - The client and server could not negotiate the desired level of security. Reason: unknown certificate:host=MYSERVER,port=1940 minor code: 4942F303 completed: No]**, it may be caused by not having the server's personal certificate in the client truststore.

To correct this problem:
- Check the client truststore to determine if the signer certificate from the server personal certificate is there. For a self-signed server personal certificate, the signer certificate is the public key of the personal certificate. For a CA signed server personal certificate, the signer certificate is the root CA certificate of the CA which signed the personal certificate.
- Add the server signer certificate to the client truststore.

javax.net.ssl.SSLHandshakeException: bad certificate: If you see a java exception stack similar to **ERROR: Could not get the initial context or unable to look up the starting context. Exiting. Exception received: javax.naming.ServiceUnavailableException: A communication failure occurred while attempting to obtain an initial context using the provider url: ″corbaloc:iiop:localhost:2809″. Make sure that the host and port information is correct and that the server identified by the provider url is a running name server. If no port number is specified, the default port number 2809 is used.Other possible causes include the network environment or workstation network configuration. [Root exception is org.omg.CORBA.TRANSIENT: CAUGHT_EXCEPTION_WHILE_CONFIGURING_SSL_CLIENT_SOCKET: JSSL0080E: javax.net.ssl.SSLHandshakeException - The client and server could not negotiate the desired level of security. Reason: bad certificate: host=MYSERVER,port=1940 minor code: 4942F303 completed: No]**, it can be caused by having a personal certificate in the client keystore used for SSL mutual authentication but not having extracted the signer certificate into the server truststore so that the server could trust it whenever the SSL handshake is made.

To verify this, check the server truststore to determine if the signer certificate from the client personal certificate is there. For a self-signed client personal certificate, the signer certificate is the public key of the personal certificate. For a CA signed client personal certificate, the signer certificate is the root CA certificate of the CA which signed the personal certificate.

To correct this problem, add the client signer certificate to the server truststore.

org.omg.CORBA.INTERNAL: EntryNotFoundException or NTRegistryImp E SECJ0070E: No privilege id configured for: error when programmatically creating a credential: If you encounter the following exception in a client application attempting to request a credential from a WebSphere Application Server using SSL mutual authentication:

ERROR: Could not get the initial context or unable to look up the starting context. Exiting. Exception received: org.omg.CORBA.INTERNAL: Trace from server: 1198777258 at host MYHOST on port 0 >>org.omg.CORBA.INTERNAL: EntryNotFoundException minor code: 494210B0 completed: No at com.ibm.ISecurityLocalObjectBaseL13Impl.PrincipalAuthFailReason.map_auth_ fail_to_minor_code(PrincipalAuthFailReason.java:99)

or a simutaneous error from the WebSphere Application Server that resembles:

[7/31/02 15:38:48:452 CDT] 27318f5 NTRegistryImp E SECJ0070E: No privilege id configured for: testuser

the cause may be that the user id sent by the client to the server is not in the server's user registry.

To confirm that this is the problem, check that an entry exists for the personal certificate which is being sent to the server. Depending on the mechanism used for user registry, look at the native operating system user ID's or LDAP server entries.

To correct this problem, add the user ID to the user registry entry (for example, operating system, LDAP directory, or other custom registry) for the personal certificate identity.

Errors in messaging (JMS API)
What kind of problem are you seeing?

- javax.jms.JMSException: MQJMS2008: failed to open MQ queue in JVM log.

If you do not see a problem that resembles yours, or if the information provided does not solve your problem, see Messaging (JMS) component troubleshooting tips. If you are still unable to resolve the problem, contact IBM support for further assistance.

javax.jms.JMSException: MQJMS2008: failed to open MQ queue in JVM log: This error can occur when the MQ queue name is not defined in the Internal JMS Server Queue Names List. This can occur if a WebSphere Application Server Queue Destination is created, without adding the Queue Name to the internal JMS Server Queue Names List.

To resolve this problem:
- Open the WebSphere Application Server Administrative Console.
- Click **Servers > Manage Application Servers >** *server_name>* **Server Components > JMS Servers**.
- Add the Queue Name to the list.
- Save the changes and restart the server.

Errors returned to client trying to send a SOAP request
What kind of problem are you seeing?

- SOAPException: faultCode=SOAP-ENV:Client; msg=Error opening socket; java.net.ConnectException: Connection refused: connect

- javax.security.cert.CertPathBuilderException: No end-entity certificate matching the selection criteria could be found.

If none of these errors match the one you see:
- Browse the target application server log files. (by default file `<installation_directory>/server_name/SystemErr.log` and `SystemOut.log` (main installation log file) for clues. See Viewing the JVM Logs for more information.
- Look up any error or warning messages in the message table.
- See the article Troubleshooting Universal Description, Discover, and Integration, Web Services and SOAP components for more information.

If you do not see a problem that resembles yours, or if the information provided does not solve your problem, contact IBM support for further assistance.

SOAPException: faultCode=SOAP-ENV:Client; msg=Error opening socket; java.net.ConnectException: Connection refused: connect: The most likely cause of this refused connection is that it was sent to the default port, 80, and an HTTP server is not installed or configured.

To verify this situation, send the message directly to the SOAP port. For example, to `<strong>http://<hostname>:9080</strong>`. If this works, there are two ways to resolve the problem:
- Continue specifying port 9080 on SOAP requests.
- If an HTTP server such as the IBM HTTP Server, iis, IPlanet, or others, is not installed, install one and then step through the WebSphere Application Server installation to install the associated plug-in component.
- If an HTTP server is installed:
  - Regenerate the HTTP plug-in configuration in the administrative console by clicking **Environment > Update WebServer Plugin**, and restart the HTTP server.
  - If the problem persists, view the HTTP server access and error logs, as well as the `<install_dir>/logs/http_plugin.log` file for more information.

javax.security.cert.CertPathBuilderException: No end-entity certificate matching the selection criteria could be found: This error usually indicates that new or updated security keys are needed. The security key files are:
- SOAPclient
- SOAPserver
- sslserver.p12

In an installed application, these files are located in: `<install_dir>/installedApps/<application_name>.ear/soapsec.war/key/`. After replacing these files, you must stop and restart the application.

To replace these files in a SOAP-enabled application that has not yet been installed:
- Expand `<application_name>.ear`.
- Expand `soapsec.war`.
- Replace the security key files in the `key/` directory.
- After you have replaced these files, install the application and restart the server.

Client program does not work
What kind of problem are you seeing?

- ActiveX client fails to display ASP files, or WebSphere Application Server resources (JSP files, servlet, or HTML pages), or both.

If you do not see a problem that resembles yours, or if the information provided does not solve your problem, contact IBM support for further assistance.

ActiveX client fails to display ASP files, or WebSphere Application Server resources (JSP files, servlet, or HTML pages),or both: A possible cause of this problem is that both IIS (for serving ASP files) and an HTTP server that supports WebSphere Application Server (such as IBM HTTP Server) are deployed on the same host. This leads to misdirected HTTP traffic if both servers are listening on the same port (such as the default port 80).

To resolve this problem, do one of the following:
- Open the IIS administrative panel, and edit the properties of the default Web server to change the port number to something other than 80;
- Install IIS and the WebSphere Application Server HTTP server on separate servers.

Troubleshooting application run-time and management problems

Select the problem you are having with running or managing deployed code for WebSphere Application Server:
- I have problems bringing up or using the administrative console.
- I have problems starting or using the wsadmin command prompt.
- My Web module or application server dies or hangs.
- I get errors trying to configure and enable security.
- The workload is not being distributed across servers.
- I can't uninstall or remove a node or application server.
- I have problems creating or using HTTP sessions.

If you do not see a problem that resembles yours, or if the information provided does not solve your problem, contact IBM support for further assistance.

Installation completes but the administrative console does not start

What kind of problem are you having?
- "Internal Server Error", "Page cannot be found", 404, or similar error trying to view administrative console.
- "Unable to process login. Please check User ID and password and try again. " error when trying to access console page.
- Directory paths in the console are garbled.

If you are able to bring up the browser page, but the console's behavior is inconsistent, error-prone, or unresponsive, try upgrading the browser you are using. Older browsers may not support the administrative console's features.

If none of these steps solves the problem, check to see if the problem has been identified and documented using the links in Diagnosing and fixing problems: Resources for learning. If you do not see a problem that resembles yours, or if the information provided does not solve your problem, contact IBM support for further assistance.

″**Internal Server Error**″, ″**Page cannot be found**″**, 404, or similar error trying to view administrative console:** If you are unable to view the administrative console, here are some steps to try:

- Verify that the application server which supports the administrative console is up and running.
  - For a "base" configuration, the administrative console is deployed by default on "server1". Before viewing the administrative console, you must:
    - Run the **startServer server1** command for Windows or **./startServer.sh server1** command for Unix from a command prompt in the *install_dir*\bin directory, or
    - Click the "start application server" link from the "first steps" panel, or
    - Start WebSphere Application Server as a service or from the Start menu, if you are using Windows.
  - If you are using the Deployment Manager (for a multi-node configuration), run the startManager command from the *Network_Deployment_install_dir*\bin directory.
  - View the SystemOut.log file for the application server or deployment manager to verify that the server supporting the administrative console has actually started.
- Check the URL you are using to view the console. By default, it is http://*server_name*:9090/admin.
- If you are browsing the console from a remote machine, try to eliminate connection, address and firewall issues by:
  - Pinging the server machine from a command prompt, using the same server name as in the URL.
  - If you have access to the server, try browsing the console locally using http://*server_name*:9090\admin.
- If you have never been able to access the administrative console, verify that the installation was successful.

″**Unable to process login. Please check User ID and password and try again.** ″ **error when trying to access console page:** This error indicates that security has been enabled for WebSphere Application Server, and the user ID or password supplied is either invalid or not authorized to access the console.

To access the console,

- If you are the administrator, use the ID defined as the security administrative ID. This ID is stored in the WebSphere Application Server directory structure in the file security.xml.
- If you are not the administrator, ask the administrator to enable your ID for the administrative console.

Directory paths in the console are garbled: If directory paths used for classpaths or resources specified in the Application Assembly Tool, configuration files, or elsewhere, appear garbled in the administrative console, it may be because the Java runtime interprets a backslash (\) as denoting a control character.

To resolve, modify Windows-style classpaths by replacing occurrences of single backslashes to two. For example, change "c:\MyFiles\MyJsp.jsp" to "c:\\MyFiles\\MyJsp.jsp".

Problems starting or using the wsadmin command
What kind of problem are you having?

- "WASX7023E: Error creating "SOAP" connection to host" or similar error trying to launch wsadmin command line utility.
- "com.ibm.bsf.BSFException: error while eval'ing Jacl expression: no such method "<command name>" in class com.ibm.ws.scripting.AdminConfigClient" returned from wsadmin command.
- WASX7022E returned from running "wsadmin -c ..." command, indicating invalid command.
- com.ibm.ws.scripting.ScriptingException: WASX7025E: String "" is malformed; cannot create ObjectName.
- "The input line is too long" error returned from the wsadmin command on a Windows platform.

If you do not see your problem here:
- If you are not able to enter wsadmin command mode, try running **wsadmin -c** "**$Help wsadmin**" for help in verifying that you are entering the command correctly.
- If you can get the wsadmin command prompt, enter **$Help help** to verify that you are using specific commands correctly.
- wsadmin commands are a superset of Jacl (Java Command Language), which is in turn a Java-based implementation of the Tcl command language. For details on Jacl syntax beyond wsadmin commands, refere to the Tcl developers' site, http://www.tcl.tk. For specific details relating to the Java implementation of Tcl, refer to http://www.tcl.tk/software/java.
- Browse the *install_dir*/logs/wsadmin.traceout file for clues.
  - Keep in mind that wsadmin.traceout is refreshed (existing log records are deleted) whenever a new wsadmin session is started.
  - If the error returned by wsadmin does not seem to apply to the command you entered, for example, you receive WASX7023E, stating that a connection could not be created to host "myhost," but you did not specify "-host myhost" on the command line, examine the properties files used by wsadmin to determine what properties are specified. If you do not know what properties files were loaded, look for the WASX7326I messages in the wsadmin.traceout file; there will be one of these messages for each properties file loaded.

If none of these steps fixes your problem, check to see if the problem has been identified and documented by looking at the available online support (hints and tips, technotes, and fixes). If you don't find your problem listed there contact IBM support.

"**WASX7023E: Error creating "SOAP" connection to host" or similar error trying to launch wsadmin command line utility:** By default, the wsadmin utility attempts to connect to an application server at startup. This is because some commands act upon running application servers. This error indicates that no connection could be established.

To resolve this problem:
- If you are not sure whether an application server is running, start it by entering **startserver** *servername* from the command prompt. If the server is already running, you will see an error similar to "ADMU3027E: An instance of the server is already running".
- If you are running a Network Deployment configuration, you will first need to start the deployment manager by running "startManager" or "startManager.sh"

from the *install_dir*/bin directory. Then you can launch wsadmin immediately to connect to the deployment manager, or start a node and application server to connect to.

- If an application server is running and you still get this error:
  - If you are running remotely (that is, on a different machine from the one running WebSphere Application Server), you must use the **-host** *hostname* option to the wsadmin command to direct wsadmin to the right physical server.
  - If you are using the -host option, try pinging the server machine from the command line from the machine on which you are trying to launch wsadmin to verify there are no issues of connectivity such as firewalls.
  - verify that you are using the right port number to connect to the WebSphere Application Server process:
    - If you are not specifying a port number (using the -port option) when you start wsadmin, wsadmin uses the default port specified in *install_dir*/properties/wsadmin.properties, property name=com.ibm.ws.scripting.port (default value =8879).
    - The port that wsadmin should send on depends on the server process wsadmin is trying to connect to.

      For a single-server installation, wsadmin attempts to connect to the application server process by default. To verify the port number:
      - Look in the file *install_dir*/config/cells/*node_name*/nodes/*node_name*/serverindex.html for a tag containing the property **serverType="APPLICATION_SERVER"**.
      - Look for an entry within that tag with the property **endPointName="SOAP_CONNECTOR_ADDRESS"**.
      - Look for a **port** property within that tag. This is the port wsadmin should send on.

      In a Network Deployment installation, wsadmin launched from the bin directory on the Network Deployment installation attempts to send requests to the deployment manager by default. To verify the port number:
      - Get the hostname of the node on which the Deployment Manager is installed.
      - Using that hostname, look in *install_dir*/config/cells/ *node_name*Network/nodes/*node_name*Manager/serverindex.html for a tag containing the property **serverType="DEPLOYMENT_MANAGER"**.
      - Within that tag, look for an entry with a property **endPointName="SOAP_CONNECTOR_ADDRESS"**.
      - Within that tag, look for a "port" property. This is the port wsadmin should send on.

"**com.ibm.bsf.BSFException: error while eval'ing Jacl expression: no such method** "**<command name>**" **in class com.ibm.ws.scripting.AdminConfigClient**" **returned from wsadmin command:** This error is usually caused by a misspelled command name. Use the **$AdminConfig help** command to get information about what commands are available. Note that command names are case-sensitive.

WASX7022E returned from running "**wsadmin -c ...**" **command, indicating invalid command:** If the command following -c appears to be valid, the problem may be caused by the fact that on Unix, using wsadmin -c to invoke a command that includes dollar signs results in the shell attempting to do variable substitution. To

confirm that this is the problem, check the command to see if it contains an unescaped dollar sign, for example: **wsadmin -c** ″**$AdminApp install ....**″.

To correct this problem, escape the dollar sign with a backslash. For example: **wsadmin -c** ″**\$AdminApp install ...**″.

com.ibm.ws.scripting.ScriptingException: WASX7025E: String ″″ **is malformed; cannot create ObjectName:** One possible cause of this error is that an empty string was specified for an object name. This can happen if you use one scripting statement to create an object name and the next statement to use that name, perhaps in an "invoke" or "getAttribute" command, but you don't check to see if the first statement really returned an object name. For example (the following samples use basic Jacl commands in addition to the wsadmin Jacl extensions to make a sample script):

```
#let's misspell "Server"
set serverName [$AdminControl queryNames type=Server,*]
$AdminControl getAttributes $serverName
```

To correct this error, make sure that object name strings have values before using them. For example:

```
set serverName[$AdminControl queryNames type=Server,*]
if {$serverName == ""} {puts "queryNames returned empty - check query argument"}
else {$AdminControl getAttributes $serverName}
```

For details on Jacl syntax beyond wsadmin commands, refer to the Tcl developers' site, http://www.tcl.tk.

″**The input line is too long**″ **error returned from the wsadmin command on a Windows platform:** This error indicates that the Windows command line limit of 1024 characters has been exceeded, probably due to a long path name used within the wsadmin.bat command. The problem can be avoided by using the Windows subst command, which allows you to map an entire path to a virtual drive. To see the syntax of the subst command, enter **help subst** from a Windows command prompt.

For example if the product resides in c:\TestEnvironment\Beta\WebSphere\AppServer, edit the file c:\TestEnvironment\Beta\WebSphere\AppServer\bin\setupCmdLine.bat as follows:

```
subst w: c:\TestEnvironment\Beta\WebSphere\AppServer

REM comment out the old line
REM SET WAS_HOME=C:\TestEnvironment\Beta\WebSphere\AppServer

SET WAS_HOME=w:

REM comment out the old line
REM SET JAVA_HOME=C:\TestEnvironment\Beta\WebSphere\AppServer\java

SET JAVA_HOME=w:\java
```

Web module or application server dies or hangs

If an application server dies, that is its process spontaneously closes, or freezes, that is, its web modules stop responding to new requests:

- If possible, isolate the problem by installing Web modules on different servers.
- View the topic (Monitoring performance with Tivoli Performance Viewer (formerly Resource Analyzer)). The performance viewer can be used to determine which resources have reached their maximum capacity, such as java

heap memory (indicating a possible memory leak) and database connections. If a particular resource appears to have reached its maximum capacity:

– Review the application code for a possible cause.

- If database connections are used and never freed, ensure that application code performs a **close()** on any opened **Connection** object within a **finally{}** block.

- If servlet engine threads in use steadily increase, review application **synchronized** code blocks for possible deadlock conditions .

- If a JVM's heap size steadily increases, review application code for memory leak opportunities, such as static (class-level) collections which cause objects to never get garbage-collected.

• As an alternative to using the Performance Viewer to detect memory leak problems, enable verbose garbage collection on the application server. This feature adds detailed statements to the JVM error log file of the application server about the amount of available and in-use memory.

1. Select **Servers > Application Servers > *server_name* > Process Definition > Java virtual machine**, and enable **Verbose Garbage Collection**.

2. Stop and restart the application server.

3. Periodically, or after the application server stops, browse the log file for garbage collection statements. Look for statements beginning "allocation failure" , which indicate that a need for memory allocation has triggered a JVM garbage collection (freeing of unused memory). Allocation failures themselves are normal and not indicative of a problem. This will be followed by a statements showing how many bytes are needed and how many are actually allocated.

   If the total amount of free and used memory keeps increasing, that is the JVM keeps allocating more memory for itself, or if the JVM becomes unable to allocate as much memory as it needs, as indicated by "bytes needed", there may be a memory leak.

• If the Performance Viewer or verbose garbage collection output indicate that the application server is running out of memory, indicating one of the following problems:

– There is a memory leak in application code which needs to be addressed. In order to pinpoint the cause of a memory leak Enable the RunHProf function in the Servers > Application Servers > server_name > Process Definition > Java virtual machine pane of the problem application server.

- In the same pane, set the **HProf Arguments** field to a value like `depth=20,file=heapdmp.txt`. This will show the exception stacks to a maximum of 20 levels, and save the output to the `<installation_root>/bin` directory in a file named `heapdmp.txt`.

- Save the settings, then stop and restart the application server.

- Re-enact the scenario or access the resource that causes the hang or crash, if possible, and then stop the applicatin server. If this is not possible, wait until the hang or crash happens again.

- Examine the file into which the heapdump was saved.

  • Search on the string, "SITES BEGIN". This finds the location of a list of Java objects in memory and the amount of memory allocated to them.

  • This list has a record for each time an allocation of memory was made in this JVM, what type of object the memory was used to instantiate, and an identifier of a trace stack, listed elsewhere in the dump, which shows the Java method in which the allocation was made.

- The list is in descending order by number of bytes allocated. Depending on the nature of the leak, the problem class should show up near the top of the list, but this is not always the case. Look for large amounts of memory or frequent instances of the same class getting instantiated thoughout the list. In the latter case, use the ID in the `trace stack` column to find allocations happening repeatedly in the same class and method.
    - Examine the source code indicated in the related trace stacks for the possibility of memory leaks.
  – The default maximum heap size of the application server needs to be increased, or
  – There is a defect in the WebSphere Application Server product which needs to be reported, or which may already be fixed in a maintenance download. Contact IBM support.
- If an application server has spontaneously died, look a for a java thread dump file. This is a file created by the JVM in the product directory structure, with a name like **javacore[number].txt**.
- Thread dumps (or "javacores") can also be forced from running applications. The way to create a thread dump differs from earlier releases of the product:
  1. Using the wsadmin command prompt, first get a handle to the problem application server: **wsadmin>set jvm** *$AdminControl completeObjectName type=JVM,process=server1,**
  2. Generate the thread dump: **wsadmin>$AdminControl invoke $jvm dumpThreads**.
  3. Look for an output file in the installation root directory with a name like **javacore.***date.time.id***.txt**
- Browse the thread dump for clues.
  – If the thread dump was not manually forced (it was created by the JVM as it closed) look for an "error" or "exception information" at the beginning of the file indicating the thread which caused it to die.
  – The thread dump contains a snapshot of each thread in the process, starting in the section labeled "Full thread dump."
    - Look for threads for which the description contains "state:R". These are threads in an active, running state when the dump was forced or the process exited.
    - Look for multiple threads in the same java application code source location. These may be an indication of a deadlock condition (multiple threads waiting on a monitor) or an infinite loop, and may indicate where in application code the problem lies.

If none of these steps fixes your problem, check to see if the problem has been identified and documented by looking at the available online support (hints and tips, technotes, and fixes). If you don't find your problem listed there contact IBM support.

Errors when trying to configure or enable security

What kind of error are you seeing?
- "LTPA password not set. validation failed" message displayed as error in the Administrative Console after enabling global security.
- "Validation failed for user [userid]. Please try again..." displayed in the Administrative Console when enabling global security.

- If you have successfully configured security (made changes, saved the configuration, and enabled security with no errors), but are now having problems accessing Web resources or the administrative console, refer to Errors or access problems after enabling security.

For general tips on diagnosing and resolving security-related problems, see the topic Troubleshooting the security component.

If you do not see a problem that resembles yours, or if the information provided does not solve your problem, contact IBM support for further assistance.

"**LTPA password not set. validation failed**" **message displayed as error in the Administrative Console after saving global security settings:** This error can be caused if, when configuring WebSphere Application Server security, "LTPA" is selected as the authentication mechanism, and the LTPA password field is not set. To resolve this problem:

- Select **Authentication Mechanism -> LTPA**.
- Complete the password and confirm password fields.
- Click OK.
- Try setting Global Security again.

"**Validation failed for user userid. Please try again...**" **displayed in the Administrative Console after saving global security settings:** This typically indicates that a setting in the User Registry configuration is not valid:

- If the user registry is LocalOS, it's likely that either the server userid and password is invalid or the server userid does not have "Act As Part of the Operating System" (for NT) or root authority (for Unix). It needs this authority in order to access the LocalOS user registry to authenticate.
- If the user registry is Lightweight Directory Access Protocol (LDAP):
  - Any of the settings that enable WebSphere Application Server to communicate with LDAP might be invalid, such as the LDAP server's userid, password, host, port, or LDAP filter. When you select **Apply** or **OK** on the Global Security panel, a validation routine connects to the registry just as it would during runtime when security is enabled. This is done in order to verify any configuration problems immediately, instead of waiting until the server restarts.
  - If the BIND DN is required, you must specify a DN instead of a short name.
  - Sometimes the LDAP server might be down during configuration. The best way to check this is to issue a command line search - LdapSearch, to search for the server ID. This way you can determine if the server is running and if the server ID is a valid entry in the LDAP.
- If the user registry is Custom, double check that your implementation is in the classpath. Also, check to see if your implementation is authenticating properly.
- Regardless of registry type, check the User Registries configuration panels to see if you can find a configuration error:
  - Go back to the User Registries configuration panels and retype the password for the server ID.
- See if there is an obvious configuration error. Double check the attributes specified.

Cannot uninstall an application or remove a node or application server
What kind of problem are you having?

- After uninstalling an application through wsadmin tool, the application continues to run and throws "DocumentIOException"
- The removeNode command does not remove the installed application from the deployment manager
- I cannot display the syntax for the removeNode command.

If none of these steps fixes your problem:
- Make sure that the application and its Web and EJB modules, are in a stopped state before uninstalling.
- If you are uninstalling or installing an application using **wsadmin**, make sure that you are using the **-conntype NONE** option to invoke **wsadmin** and enable local mode. To use the **-conntype NONE** option, stop the hosting application server before uninstalling the application.
- Check to see if the problem has been identified and documented by looking at the available online support (hints and tips, technotes, and fixes).
- If you don't find your problem listed there contact IBM support

After uninstalling application through the wsadmin tool, the application throws "**DocumentIOException**": If this exception occurs after the application was uninstalled using wsadmin with the -conntype NONE option:
- Restart the server or,
- Rerun the uninstall command without the -conntype NONE option.

The removeNode command does not remove the installed application from the deployment manager: If the applications were installed indirectly using the **addNode** program with the **-installapp** option, then removeNode will not uninstall them, since they may be in use by other nodes. These applications must be explicitly uninstalled, for example through the administrative console.

I cannot display the syntax for the removeNode command: Unlike the addNode command, the removeNode command is valid with no parameters, so executing it will execute the operation, that is, remove the node, without displaying the command syntax.

To see the valid options for removeNode, execute removeNode -? or removeNode -help.

Workload not getting distributed or errors setting up multiserver environment
What kind of problem are you seeing?
- Web (HTTP) requests do not get distributed to all servers.
- Enterprise bean requests don't get distributed to all servers.
- Enterprise bean requests are not distributed evenly.
- After creating and then starting a cluster, the cluster will not start and logs show that the servers are not found.
- One or more nodes don't show up in the administrative console.
- When some servers fail, enterprise bean requests are still sent to the bad server (failover fails).
- The addNode command fails.
- Application files are not getting copied to all nodes.
- After downloading the Network Deployment plugin to my system, my server does not start.

- Stopped or hung servers do not share the workload after being restored.

If none of these steps fixes your problem:
- Browse the JVM logs of the problem deployment manager and application servers, and
  - Look up any error messages by selecting the **Quick Reference** view of this InfoCenter and expanding **Messages** in the navigation tree.
  - Use the Log Analyzer to browse and analyze the service log (`activity.log`) of the deployment manager and any nodes encountering problems. View the `activity.log` files in both `<NetworkDeployment_installation_root>/logs` and `<ApplicationServer_installation_root>/logs`.
  - If Java exceptions appear in the log files, try to determine the actual subcomponent directly involved in the problem by examining the trace stack and looking for a WebSphere Application Server-related class near the top of the stack (with names beginning `com.ibm.websphere` or `com.ibm.ws`) which threw the exception. If appropriate, review the steps for troubleshooting the appropriate subcomponent under Troubleshooting by component: what is not working?

    For example, if the exception appears to have been thrown by a class in the `com.ibm.websphere.naming` package, review the topic, Naming Services Component troubleshooting tips in this InfoCenter.
- Ensure that all the machines in your configuration have TCP/IP connectivity to each other by running the ping command:
  - From each physical server to the Deployment Manager, and
  - From the Deployment Manager to each physical server.
- Although the problem is happening in a clustered environment, the actual cause may be only indirectly related, or unrelated, to clustering. Be sure to investigate all relevant possibities:
  - If an enterprise bean on one or more servers is not serving requests, review the topics Cannot access an enterprise bean from a servlet, jsp, standalone program, or other client and Cannot access an object hosted by WebSphere Application Server from a servlet, jsp file, or other client.
  - If problems seem to appear after enabling security, review the topic Errors or access problems after enabling security.
  - If one or more application server stops responding to requests, or spontaneously dies (its process closes), review the topic Web module or application server dies or hangs.
  - If SOAP requests are not being served by some or all servers, review the topic Errors returned to client trying to send a SOAP request.
  - If you have problems installing or deploying an application on servers on one or more nodes, review the topic, Troubleshooting code deployment and installation problems.
- If your topology consists of a Windows-based Deployment Manager with Unix-based servers, browse any recently-updated `.xml` and `.policy` files on the Unix platforms using vi to ensure that Control-M characters haven't been inadvertently inserted in the files. To avoid this problem in the future edit these files using vi on the Unix platforms to avoid inserting these characters.
- Check the steps for troubleshooting the Workload Management component.
- check to see if the problem is identified and documented by looking at the available online support (hints and tips, technotes, and fixes). If you don't find your problem listed there contact IBM support.

Web (HTTP) requests don't get distributed to all servers: If HTTP requests are not being distributed to all servers:

- Check your PrimaryServers list. The plugin will load balance across all servers that are defined in the PrimaryServers list if affinity has not been established. If you do not have a PrimaryServers list defined then the plugin will load balance across all servers defined in the cluster if affinity has not been established. In the case where affinity has been established, the plugin should go directly to that server for all requests within the same HTTP session.
- If some servers are servicing requests and one or more others are not, try accessing a problem server directly to verify that it works, apart from workload management issues. If that does not work:
  - Use the Administrative Console to ensure that the affected server is running.
  - See the article Web resource (JSP, servlet, html file, image, etc) will not display for more information.
- See the article HTTP plugin component troubleshooting tips for more information.
- Check the steps for diagnosing workload management issues in Troubleshooting the Workload Management component.

Enterprise bean requests don't get distributed to all servers: If a client cannot reach a server in a cluster thought to be reachable, a server may have been marked unusable, or is down. To verify this:

- Use the Administrative Console to verify that the server is started. Try starting it, or if started restart it.
- Browse the administrative console and verify that the node that runs the server having the problem appears. If it does not:
  - Review the steps for adding a node to a cluster.
  - Review the steps under "One or more nodes don't show up in the admin console."
- If possible, try accessing the enterprise bean directly on the problem server to see if there is a problem with TCP/IP connectivity, application server health, or other problem not related to workload management. If this fails, review the topic Cannot access enterprise bean from a servlet, jsp, standalone program , or other client.
- Check the steps for diagnosing workload management issues in Troubleshooting the Workload Management component.

Enterprise bean requests are not distributed evenly: There are a number of possible reasons for this behavior, which generally fall into one or more of these categories:

- improper configuration
- environment issues such as the availability of servers or applications.
- a large numbers of requests that involve transactional affinity, or
- a small number of clients

Work load management of WebSphere Application Server is based on a round robin scheme of request distribution. This results in balance being determined by numbers of requests rather than by any other measure. A true balance problem is determined by comparing the number of requests processed by each member of the cluster with the weights that have been set for each of those members. This is done by following the step in the Troubleshooting the Workload Management component.

- When the percentage of requests that arrive for each member of the cluster is consistent with the weights then further analysis of the application is required to determine the cause for the workload being imbalanced even when the number of requests is balanced.
- When the number of **numIncomingNonWLMObjectRequests** is not balanced among the members of the cluster and is large in relation to the **numIncomingRequests** then the reason for the imbalance is the non-distributable components installed on the members of the cluster. A modification to the configuration will yield a more balanced environment.
- When the number of n**umIncomingStrongAffinityRequests** is not balanced among the members of the cluster and is large in relation to the **numIncomingRequests** then the reason for the imbalance is the requests which are invoked within a transaction. These can be reduced by installing the objects involved within a transaction within the same cluster.

After creating and then starting a cluster, the cluster will not start and logs show that the servers are not found: This error can occur when the configuration is not synchronized from the deployment manager to a node. If "auto synchronization" is enabled, wait until the synchronization has had a chance to run. If you are using manual synchronization, explicitly request a sync to each node on the cluster.

To determine whether synchronization has taken place, look at the configuration on the node machines using the administrative console and verify that the new cluster members are defined on each node.

One or more nodes do not show up in the administrative console: This can occur when there is a basic connectivity problem between the deployment manager server and other servers in the topology. To determine whether this is the problem, look for the file serverindex.xml in the deployment manager's directory structure.
- If the problem node does not appear in the list, review the steps for adding a node to the cluster.
- If the problem node does appear in the list:
  - From the deployment manager server, ping the server name as it appears in the list. If the ping command shows no communiction, verify that the hostname is correct in the list, and correct it if necessary, then restart the deployment manager.
  - If the name that appears in the list is the short name, ping the fully qualified network name. If the corrected name works, update the list and restart the deployment manager.
  - If the problem server uses Dynamic Host Configuration Protocol (DHCP), try replacing the logical hostname with the IP address and restart the deployment manager. If this resolves the problem be aware that you will need to change serverindex.xml every time the problem server's address changes, potentially every time the problem machine is rebooted. To avoid this problem, consider assigning a static IP address to the server.
- If you still cannot establish communication between the servers, contact your network administrator to resolve the problem, and restart the deployment manager after the problem is corrected.

When some servers fail, Enterprise bean requests still sent to the bad server (failover fails): Some possible causes of this problem are:
- The client may have been in a transaction with an enterprise bean on the server that went down. Check the JVM logs of the application server hosting the problem enterprise bean instance. If a request is returned with **CORBA**

SystemException COMM_FAILURE org.omg.CORBA.completion_status.COMPLETED_MAYBE, this may be working as designed. The design is to let this particular exception flow back to the client, since the transaction may have completed. Failing over this request to another server could result in this request being serviced twice.

- If the requests sent to the servers come back to the client with any other exceptions consistently, it may be that no servers are available. In this case, follow the resolution steps as outlined in Troubleshooting the Workload Management component.

The addNode command fails: This error can occur when the deployment manager's Domain Name Server (DNS) configuration is not properly set up. The default installation on Linux uses the loopback address (127.0.0.1) as the default host address. To verify that this is the problem, query the hostname of the suspect machine. If it returns localhost 127.0.0.1, or if file transfer traces at the node show the node trying to upload files to a URL that includes 127.0.0.1, the node has an incorrect DNS configuration.

To correct this problem, update the /etc/hosts file or the name service configuration file (/etc/nsswitch.conf) to query the Domain Name Server or Network Information Server (NIS) before searching hosts.

Application files are not getting copied to all nodes: In the WebSphere Application Server Network Deployment environment, the application binary files are transferred to the individual nodes where applications are supported as part of the **node sync** operation. During node sync, application files are only propogated if their deployment descriptors specify **enableDistribution=true**. This flag is specified as part of the application installation procedure in the administrative console, and is stored as a property in the d<install_dir>/config/cells/<cell_name>/applications/ <application_name>/deployment.xml file.

To confirm that this is the cause, check to see whether the enableDistribution flag is set. If it is already set to true, ensure that the target node is configured to run auto file synchronization.

If both of these settings are correct and the problem persists, manually perform an explicit synchronization. If the application files still do not appear in the installation directory, use the EARExpander tool (located in `$WAS_ROOT/bin`) to expande the ear file from the repository to the installation destination. On the remote nodes the repository should appear in `config/cells/<cell_name>/applications/<application_name>.ear/directory`.

After downloading the Network Deployment plugin to my system, my server does not start: If you experience this situation, the most likely cause is that the transport paths in the plugin must be modified to work in your environment. See Example: Manually editing transport settings in the server.xml file for information on how to modify these settings.

Stopped or hung servers do not share the workload after being restored: This error occurs when the servers that were unavailable are not recognized by the Workload Management component after they are restored. There is an "unusable" interval determined by the property `com.ibm.websphere.wlm.unusable.interval` during which the WLM will wait to send to a server that has been marked unusable. By default this is 15 minutes.

You can confirm that this is the problem by ensuring that the servers that were down are now up and capable of servicing requests. Then wait for the unusable interval to elapse before checking to determine whether failback occurred.

Problems creating or using HTTP sessions

Note: To view and update the Session Manager settings discussed here, use the administrative console. Select the application server that hosts the problem application, then under **Additional properties**, select **Web Container**, then **Session manager**.

What kind of problem are you having?

- HTTP Sessions are not getting created, or are lost between requests.
- HTTP Sessions are not persistent (session data lost when application server restarts, or not shared across cluster).
- Session is shared across multiple browsers on same client machine.
- Session is not getting invalidated immediately after specified Session timeout interval.
- Unwanted sessions are being created by jsps.

If your problem is not described here, or none of these steps fixes the problem:

- Review Troubleshooting the HTTP Session Manager for general steps on debugging Session-manager related problems.
- Review Managing HTTP sessions for information on how to configure the Session manager, and best practices for using it.
- Check to see if the problem has been identified and documented by looking at the available online support (hints and tips, technotes, and fixes).
- If you don't find your problem listed there contact IBM support.

HTTP Sessions are not getting created, or are lost between requests: By default, the Session Manager uses cookies to store the session ID on the client between requests. Unless you intend to avoid cookie-based session tracking, ensure that cookies are flowing between WebSphere Application Server and the browser:

- Make sure the **Enable cookies** checkbox is checked under the **Session tracking Mechanism** property.
- Make sure cookies are enabled on the browser you are testing from or from which your users are accessing the application.
- Check the Cookie domain specified on the SessionManager (to view the or update the cookie settings, in the **Session tracking mechanism->enable cookies** property, click **Modify**).
  - For example, if the cookie domain is set as ".myCom.com", resources should be accessed using that domain name, e.g. http://www.myCom.com/myapp/servlet/sessionservlet.
  - If the domain property is set, make sure it begins with a dot (.). Certain versions of Netscape do not accept cookies if domain name doesn't start with a dot. Internet Explorer honors the domain with or without a dot. For example, if the domain name is set to *mycom.com*, change it to *.mycom.com* so that both Netscape and Internet Explorer honor the cookie.
- Check the **Cookie path** specified on the SessionManager. Check whether the problem url is hierarchially below the Cookie path specified. If not correct the Cookie path.
- If the `Cookie maximum age` property is set, ensure that the client (browser) machine's date and time is the same as the server's, including the time zone. If

the client and the server time difference is over the "Cookie maximum age" then every access would be a new session, since the cookie will "expire" after the access.

- If you have multiple web modules within an enterprise application that track sessions:
  - If you want to have different session settings among web modules in an enterprise application, ensure that each web module specifies a different cookie name or path, or
  - If Web modules within an enterprise application use a common cookie name and path, ensure that the HTTP session settings, such as Cookie maximum age, are the same for all Web modules. Otherwise cookie behaviour will be unpredictable, and will depend upon which application creates the session. Note that this does not affect session data, which is maintained separately by Web module.
- Check the cookie flow between browser and server:
  1. On the browser, enable "cookie prompt". Hit the servlet and make sure cookie is being prompted.
  2. On the server, enable SessionManager trace. Enable tracing for the HTTP Session Manager component, by using the trace specification "com.ibm.ws.webcontainer.httpsession.*=all=enabled". After trace is enabled, excericise your session-using servlet or jsp, then follow the instructions for dumping and browsing the trace output .
  3. Access the session servlet from the browser.
  4. The browser will prompt for the cookie; note the jsessionid.
  5. Reload the servlet, note down the cookie if a new cookie is sent.
  6. Check the session trace and look for the session id and trace the request by the thread. Verify that the session is stable across web requests:
     - Look for **getIHttpsession(...)** which is start of session request.
     - Look for **releaseSesson(..)** which is end of servlet request.
- If you are using URL rewriting instead of cookies:
  - Ensure there are no static HTML pages on your application's navigation path.
  - Ensure that your servlets and jsp files are implementing URL rewriting correctly. For details and an example see Session tracking options.
- If you are using SSL as your session tracking mechanism:
  - Ensure that you have SSL enabled on your IHS or iPlanet http server.
  - Review Session tracking with SSL information.
- If you are in a clustered (multiple node) environment, ensure that you have session persistence enabled.

HTTP Sessions are not persistent: If your HTTP sessions are not persistent, that is session data is lost when the application server restarts or is not shared acrss the cluster:

- Check the Datasource.
- Check the SessionManager's Persistence Settings properties:
  - If you intend to take advantage of Session Persistence, verify that Persistence is set to **Database** or **Memory to Memory Replication**.
  - If you are using **Database-based persistence**:
    - Check the jndi name of the datasource specified correctly on SessionManager.
    - Specify correct userid and password for accessing the database.

Note that these settings have to be checked against the properties of an existing Data Source in the admin console. The Session Manager does not automatically create a session database for you.

- The Datasource should be non-JTA, i.e. non XA enabled.
- Check the JVM logs for appropiate database error messages.
- With DB2, for row sizes other than 4k make sure specified row size matches the db2 page size. Make sure tablespace name is specified correctly.

– If you are using **memory-based persistence**, available in a network-deployment (multiple application server) configuration only:

- Review Memory-to-memory replication and Configuring for Memory-to-memory replication.
- Review the **Internal Replication Domains properties** of your Session manager.

Session is shared across multiple browsers on same client machine: This behaviour is browser-dependent. It varies between browser vendors, and also may change according to whether a browser is launched as a new process or as a subprocess of an existing browser session (for example by hitting Ctl-N on Windows).

The Cookie maximum age property of the Session Manager also affects this behaviour, if cookies are used as the session-tracking mechanism. If the maximum age is set to some positive value, all browser instances share the cookies, which are persisted to file on the client for the specifed maximum age time.

Session is not getting invalidated immediately after specified Session timeout interval: The SessionManager invalidation process thread runs every x seconds to invalidate any invalid sessions, where x is determined based on the Session timeout interval specified in the Session manager properties. For the default value of 30 minutes , x is around 300 seconds. In this case, it could take up to 5 minutes (300 seconds) beyond the timeout threshold of 30 minutes for a particular session to become invalidated.

Unwanted sessions are being created by jsps: As required by the Java Server Page specification, jsps by default perform a request.getSession(true), so that a session is created if none exists for the client. To prevent jsps from creating a new session, set the session scope to false in the jsp using the page directive as follows:

```
<% @page session="false" %>
```

Troubleshooting by component: what is not working?

This section provides troubleshooting information based on the task you were trying to accomplish when the problem occurred. To find more information about your problem, select a task category from the list below.

If you do not see a task that resembles yours, or if the information provided does not solve your problem, contact IBM support for further assistance.

Installation component troubleshooting tips

If you are having problems installing the WebSphere Application Server, follow these steps to resolve the problem:

• If possible, follow the steps outlined in Troubleshooting the installation.

- Browse the relevant log files for clues:
  - The main installation log file: `<install_dir>/log.txt`.
  - IBM Http Server log: `<install_dir>/ihs.log`.
  - The log file produced when the default application `.ear` file is installed is: `<install_dir>/logs/installDefaultApplication.log`.
- Ensure that you have installed the correct level of dependent software, such as operating system version and revision level, by reviewing `http://www.ibm.com/software/webservers/appserv/doc/latest/prereq.html`.

If none of these steps solves the problem, check to see if the problem is identified and documented using the links in Diagnosing and fixing problems: Resources for learning. If you do not see a problem that resembles yours, or if the information provided does not solve your problem, contact IBM support for further assistance.

Migration utility troubleshooting tips

If you are encounter problems migrating an application from a previous version of WebSphere Application Server to Version 5.0:
- Look for these log files and browse them for clues:
  - `<install_dir>/logs/WASPostUpgrade.<time stamp>.log`
  - `<migration_backup_dir>/WASPreUpgrade.<time stamp>.log`
  - `<install_dir>/logs/clientupgrade.<time stamp>.log`
- Look for **MIGR0259I: Completed successfully** or **MIGR0271W: Completed with warnings** in the `<migration_backup_dir>/WASPreUpgrade.<time stamp>.log`, `<migration_backup_dir>/WASPreUpgrade.<time stamp>.log`, or `<install_dir>/logs/clientupgrade.<time stamp>.log`.

  If **MIGR0286E: Completed with errors.** appears, attempt to correct any problems based on the error messages that appear in the log file. After correcting any errors, rerun the command from the bin directory of the product installation root. If the errors persist, rerun the command with trace enabled.
- To generate more detailed messages when running the migration tools, enable tracing:
  - when running the WASPreUpgrade or WASPostUpgrade tools, add the following strings when you invoke them: **-traceString** ″***=all=enabled**″ **-traceFile** *migration_backup_dir/filename*.
  - when running ClientUpgrade, add the following strings to the command line when you invoke it: **-traceString** ″***=all=enabled**″ **-traceFile** *install_dir/logs/filename*.
- Open the log analyzer on the service log of the server which is hosting the resource you are trying to access and use it to browse error and warning messages.
- With WebSphere Application Server running, run the **dumpNameSpace** on Windows or **dumpNameSpace.sh** command on Unix, and pipe, redirect, or "more" the output so that it can be easily viewed. This command results in a display of all objects in WebSphere Application Server's namespace, including the directory path and object name.
- If the object a client needs to access does not appear, use the administrative console to verify that:
  - The server hosting the target resource is started.
  - The web module or EJB container hosting the target resource is running.
  - The JNDI name of the target resource is properly specified.

- To view detailed information on the runtime behavior of WebSphere Application Server's Naming service, enable trace on the following components and review the output:
  - com.ibm.ws.naming.*
  - com.ibm.websphere.naming.*

If none of these steps solves the problem, see Troubleshooting migration problems for tips on specific migration problems. If none of these match your problem, check to see if the problem is identified and documented using the links in Diagnosing and fixing problems: Resources for learning. If you do not see a problem that resembles yours, or if the information provided does not solve your problem, contact IBM support for further assistance.

Administration and Administrative Console troubleshooting tips

In the WebSphere Application Server, administrative functions are supported by the

- application server (such as "server1") in the base installation, or
- the Deployment Manager in the Network Deployment configuration.

This process must be running in order to use the administrative console. The wsadmin command line utility has a local mode that can be used to perform administrative functions even when the server is not running.

If you have problems starting or using the administrative console or wsadmin utility, verify that the supporting server process started and is healthy.

- For a base installation, look at the files:
  - *installation_directory*/logs/server/startServer.log for this message indicating that the server started successfully: **ADMU3000I: Server server1 open for e-business; process id is** *nnnn.*.
  - *install_dir*/logs/*server*/SystemOut.log for this message indicating that the server started successfully: **WSVR0001I: Server** *server* **open for e-business**.
- For a Network Deployment installation, look at the files:
  - *install_dir*/logs/dmgr/startServer.log for this message indicating that the server started successfully: **ADMU3000I: Server server1 open for e-business; process id is** *nnnn.*.
  - *install_dir*/logs/dmgr/SystemOut.log for this message indicating that the server started successfully: **WSVR0001I: Server** *server* **open for e-business**.
- Look up any error messages in these files in the message reference table. Select the **Quick Reference** view in this InfoCenter, then click **Messages**.
- A message like **WASX7213I: This scripting client is not connected to a server process** when trying to start wsadmin indicates that either the server process is not running, the host machine where it is running is not accessible, or that the port or server name used by wsadmin is incorrect.
- Verify that you are using the right port number to communicate with the admin console or wsadmin server using the following steps:
  - Look in the SystemOut.log.
  - The line **ADMC0013I: SOAP connector available at port** *nnnn* indicates the port the server is using to listen for wsadmin functions.
  - The property **com.ibm.ws.scripting.port** in the file *install_dir*/properties/wsadmin.properties controls the port used by wsadmin

to send requests to the server. If it is different from the value shown in the SystemOut.log file, either change the port number in this file, or specify the correct port number when starting wsadmin by using the **-port** *port_number* property on the command line.

– The message **SRVE0171I: Transport http is listening on port** *nnnn* **(default 9090)** indicates the port the server uses to listen for administrative console requests. If it is different than the one specified in the URL for the admin console, change the URL in the browser to the correct value. The default value is http://localhost:9090/admin.

- Use the TCP/IP **ping** command to test that the hostname where the application server or Deployment manager is executing is reachable from the system where the browser or wsadmin program are being used. If you are able to ping this hostname, this indicates that there are no firewall or connectivity issues.

- if the host where the application server or DeploymentManager is running is remote to the machine from which the client browser or wsadmin command is running, ensure that

  – the hostname in the browser URL for the console is correct, or

  – the **-host** *hostname* option of the wsadmin command is being used to direct wsadmin to the right server.

If none of these steps solves the problem, See if the specific problem you are having is addressed in the topic Installation completes but the administrative console does not start in this InfoCenter. Check to see if the problem has been identified and documented using the links in Diagnosing and fixing problems: Resources for learning. If you do not see a problem that resembles yours, or if the information provided does not solve your problem, contact IBM support for further assistance.

Application Assembly Tool troubleshooting tips

If you are having problems installing the WebSphere Application Server Application Assembly Tool (AAT), follow these steps:

- If a problem occurs using this component, the first thing to do is to enable the printing of messages and exceptions to the screen.

  – Modify the `assembly.bat` file located in the bin directory of the product installation. Change the statement ″`start javaw`″ to just ″`java`″.

  – Restart the AAT and a hanging command prompt window will appear through the lifetime of the Java process and display messages and exceptions.

  – Look up any error or warning messages you see in the message reference table.

- With a problem application open in the AAT, use the **Verify** menu command. This command will go through all components of the application and validate them for any XML errors or invalid entries such as missing fields, invalid bean or class references.

- To verify the integrity of an EAR (Enterprise Application Resource) file, expand it manually (outside of the AAT) by running the WebSphere Application Server `<install_root>\bin\EARExpander.bat` or `EARExpander.sh` file and supplying the name of the EAR file as a parameter. Browse the directory structure of the expanded EAR file to see if contains all the expected files.

  Here is an example using the Windows command prompt: `EARExpander -ear my.ear -expandDir c:\tmp\myear -operation expand`

- Contact the developer of the EAR file or its component files and ensure that they comply with J2EE specification level 1.3 and that any enterprise beans it contains conform to the EJB 2.0 Specification level.

If none of these steps solves the problem, check to see if the problem is identified and documented using the links in Diagnosing and fixing problems: Resources for learning. If you do not see a problem that resembles yours, or if the information provided does not solve your problem, contact IBM support for further assistance.

Web Container troubleshooting tips

If you are having problems starting a Web module, or accessing resources within a particular Web module:

- View the JVM logs and process logs for the application server which hosts the problem Web modules, and look for messages in the JVM output file which indicate that the web module has started successfully. You should see messages similar to the following:

```
WebContainer  A SRVE0161I: IBM WebSphere Application Server - Web Container.
Copyright IBM Corp. 1998-2002
WebContainer  A SRVE0169I: Loading Web Module: [module_name]
ApplicationMg A WSVR0221I: Application started: [application_name]
HttpTransport A SRVE0171I: Transport http is listening on port [port_number]
[server_name] open for e-business in [install_root]/log/[server_name]/SystemOut.log
```

- For specific problems that can cause servlets, html files, and jsp files not to be served, see Web resource (JSP, servlet, html file, image, etc) will not display.
- Use the Log Analyzer tool to browse the service log (activity.log) file for clues.
- For a detailed trace of the runtime behaviour of the Web container, enable trace for the component com.ibm.ws.webcontainer.*.

If none of these steps fixes your problem, check to see if the problem has been identified and documented by looking at the available online support (hints and tips, technotes, and fixes). If you don't find your problem listed there contact IBM support.

HTTP plugin component troubleshooting tips

If you are having problems with the HTTP plugin component - the component which sends requests from your HTTP server, such as IBM HTTP Server, Apache, Domino, iPlanet, or IIS, to the Websphere Application Server, try these steps:

- Review the file <install_dir>/logs/http_plugin.log for clues. Look up any error or warning messages in the message table.
- Review your HTTP server's error and access logs to see if the HTTP server is having a problem:
  - IBM HTTP Server and Apache: access.log and error.log.
  - Domino web server: httpd-log and httpd-error.
  - iPlanet: access and error.
  - IIS: <timedatestamp>.log.

If these files don't reveal the cause of the problem, follow these additional steps.

Plugin Problem Determination Steps

The plugin provides very readable tracing which can be beneficial in helping to figure out the problem. By setting the **LogLevel** attribute in the config/plugin-cfg.xml file to **Trace**, you can follow the request processing to see what is going wrong. At a high level:

1. The plugin gets a request.
2. The plugin checks the routes defined in the plugin-cfg.xml file.
3. It finds the server group.
4. It finds the server.
5. It picks the transport protocol, usually HTTP.
6. It sends the request.
7. It reads the response.
8. It writes it back to the client.

You can see this very clearly by reading through the trace for a single request:

- The first step is to determine if the plugin has loaded into the HTTP server successfully.
  - Check to make sure thehttp_plugin.log has been created.
  - If it has, look in it to see if any error messages indicate some sort of failure that took place during plugin initialization. If no errors are found look for the following stanza, which indicates that the plugin started normally. Ensure that the timestamps for the messages correspond to the time you started the webserver:
    ```
    --------------------System Information--------------------
    Bld date: Jul  3 2002, 15:35:09
    Webserver: IIS
    Hostname = SWEETTJ05
    OS version 4.0, build 1381, 'Service Pack 6'
    -------------------------------------------------------------
    ```
  - Some common errors are:

    lib_security: loadSecurityLibrary: Failed to load gsk library
    > The GSK did not get installed or the installation is corrupt. If the GSK did not get installed you can determine this by searching for the file gsk5ssl.dll on all drives for Win32 or see if there are any libgsk5*.so files in /usr/lib on Unix. Try reinstalling the plugin to see if you can get the GSK to install in order to fix this.

    ws_transport: transportInitializeSecurity: Keyring wasn't set
    > The HTTPS transport defined in the configuration file was prematurely terminated and did not contain the Property definitions for the keyring and stashfile. Check your XML syntax for the line number given in the error messages that follow this one to make sure the Transport element contains definitions for the keyring and stashfiles before it is terminated.

  - If thehttp_plugin.log is not created, check the webserver error log to see if any plugin related error messages have been logged there that indicate why the plugin is failing to load. Typical causes of this can include failing to correctly configure the plugin with the Webserver environment. Check the documentation for configuring the Webserver you are trying to use with the Webserver plugin.
- Determine whether there are network connection problems with the plugin and the various app servers defined in the configuration. Typically you will see the following message when this is the case:

ws_common: websphereGetStream: Failed to connect to app server, OS err=%d

Where %d is an OS specific error code related to why the connect() call failed. This can happen for a variety of reasons.

– Ping the machines to make sure they are properly connected to the network. If the machines can't be pinged then there is no way the plugin will be able to contact them. Possible reasons for this include:

- Firewall policies limiting the traffic from the plugin to the app server.

- The machines are not on the same network.

– If you are able to ping the machines then the likely cause of the problem is that the port is not active. This could be because the application server or cluster has not been started or the application server has gone down for some reason. You can test this by hand by trying to telnet into the port that the connect() is failing on. If you cannot telnet into the port the app server is not up and that problem needs to be resolved before the plugin will be able to connect() successfully.

• Determine whether other activity on the machines where the servers are installed is impairing the server's ability to service a request. Check the processor utilization as measured by the task manager, processor ID, or some other outside tool to see if it:

– Is not what was expected.

– Is erratic rather than a constant.

– Shows that a newly added member of the cluster is not being utilized.

– Shows that a failing member that has been fixed is not being utilized.

• Check the administrative console to ensure that the application servers are started. View the administrative console for error messages or look in the JVM logs.

• In the administrative console, select the problem application server and view its installed applications to verify that they are started.

If none of these steps solves the problem:

• For specific problems that can cause web pages and their contents not to display, see Web resource (JSP, servlet, html file, image, etc) will not display in this InfoCenter.

• Check to see if the problem has been identified and documented using the links in Diagnosing and fixing problems: Resources for learning.

• If you do not see a problem that resembles yours, or if the information provided does not solve your problem, contact IBM support for further assistance.

HTTP session manager troubleshooting tips

If you are having problems creating or using HTTP sessions with your Web application hosted by WebSphere Application Server, here are some steps to take:

• See HTTP session aren't getting created or are getting dropped to see if your specific problem is discussed.

• View the JVM logs for the application server which hosts the problem application:

– first, look at messages written while each application is starting. They will be written between the following two messages:

```
Starting application: <application>
.....................
Application started: <application>
```

- Within this block, look for any errors or exceptions containing a package name of com.ibm.ws.webcontainer.httpsession. If none are found, this is an indication that the session manager started successfully.

- Error ″**SRVE0054E: An error occurred while loading session context and Web application**″ indicates that SessionManager didn't start properly for a given application.

- Look within the logs for any Session Manager related messages. These messages will be in the format SESNxxxxE and SESNxxxxW for errors and warnings, respectively, where xxxx is a number identifying the precise error. Look up the extended error definitions in the Session Manager message table.

- Use the Log Analyzer tool to browse the service log (activity.log) file for clues.
- See Best practices for using HTTP Sessions.
- To dynamically view the number of sessions as a Web application is running, enable performance monitoring for HTTP sessions. This will give you an indication as to whether sessions are actually being created.
  - To learn how to enable HTTP session monitoring, see (Enabling data collection through the administrative console).
  - To learn how to view the http session counters as the application runs, see (Monitoring performance with Tivoli Performance Viewer (formerly Resource Analyzer)).
- Alternatively, a special servlet can be invoked that displays the current configuration and statistics related to session tracking. This servlet has all the counters that are in performance monitor tool and has some additional counters.
  - Servlet name: **com.ibm.ws.webcontainer.httpsession.IBMTrackerDebug**.
  - It can be invoked from any web module which is enabled to serve by class name. For example, using default_app, **http://localhost:9080/servlet/com.ibm.ws.webcontainer.httpsession.IBMTrackerDebug**.
  - If you are viewing the module via the serve-by-class-name feature, be aware that it may be viewable by anyone who can view the application. You may wish to map a specific, secured URL to the servlet instead and disable the serve-servlets-by-classname feature.
- Enable tracing for the HTTP Session Manager component:
  - Use the trace specification **com.ibm.ws.webcontainer.httpsession.*=all=enabled**. Follow the instructions for dumping and browsing the trace output to narrow the origin of the problem.
  - If you are using persistent sessions based on memory replication, also enable trace for **com.ibm.ws.drs.***.
- If you are using **database-based persistent sessions**, look for problems related to the **data source** the Session Manager relies on to keep session state information. For details on diagnosing database related problems see Errors accessing a datasource or connection pool

If none of these steps fixes your problem, check to see if the problem has been identified and documented by looking at the available online support (hints and tips, technotes, and fixes). If you don't find your problem listed there contact IBM support.

Naming services component troubleshooting tips

"Naming" is a J2EE service which publishes and provides access to resources such as connection pools, enterprise beans, message listeners, etc, to client processes. If you have problems in accessing a resource which otherwise appears to be healthy, the naming service might be involved. To investigate problems with the WebSphere Application Server Naming service:

- Browse the JVM logs for the server which is hosting the resource you are trying to access. Messages starting with NMSV are related to the Naming Service.
- Open the Log Analyzer on the service log of the server which is hosting the resource you are trying to access and use it to browse error and warning messages.
- With WebSphere Application Server running, run the dumpNameSpace command for Windows systems, or the dumpNameSpace.sh command for Unix systems, and pipe, redirect, or "more" the output so that it is easily viewed. This command results in a display of all objects in the WebSphere Application Server namespace, including the directory path and object name.
- If the object a client needs to access does not appear, use the administrative console to verify that:
  - The server hosting the target resource is started.
  - The Web module or EJB container, if applicable, hosting the target resource is running.
  - The jndi name of the target resource is correct and updated.
  - If the problem resource is remote, that is, not on the same node as the Name Server node, that the jndi name is fully qualified, including the host name. This is especially applicable to Network Deployment configurations
- View detailed information on the run time behavior of the WebSphere Application Server Naming service by enabling trace on the following components and reviewing the output:
  - com.ibm.ws.naming.*
  - com.ibm.websphere.naming.*
- If you see an exception that appears to be CORBA related ("CORBA" appears as part of the exception name) look for a naming-services-specific CORBA minor code, further down in the exception stack, for information on the real cause of the problem. For a list of naming service exceptions and explanations, see the class com.ibm.websphere.naming.WsnCorbaMinorCodes in the Javadoc.

If none of these steps solve the problem:

- For specific problems that can cause access to named object hosted in WebSphere Application Server to fail, see Cannot look up an object hosted by WebSphere Application Server from a servlet, jsp, or other client.
- Check to see if the problem has been identified and documented using the links in Diagnosing and fixing problems: Resources for learning.
- If you do not see a problem that resembles yours, or if the information provided does not solve your problem, contact IBM support for further assistance.

Messaging (JMS) component troubleshooting tips

If you are having problems deploying or executing applications which use the WebSphere Application Server messaging capabilities, review these articles in the WebSphere Application Server InfoCenter:

- "Troubleshooting message-driven beans" (not in this document)
- "Troubleshooting transactions" (not in this document)

If none of these steps solves the problem, check to see if the problem has been identified and documented using the links in Diagnosing and fixing problems: Resources for learning. If you do not see a problem that resembles yours, or if the information provided does not solve your problem, contact IBM support for further assistance.

Universal Discovery, Description, and Integration, Web Service, and SOAP components troubleshooting tips

If you are having problems deploying or executing applications that use WebSphere Application Server Web Services, Universal Discovery, Description, and Integration (UDDI), or SOAP, try these steps:

- Review the troubleshooting documentation for messaging in this InfoCenter:
  - WSIF troubleshooting tips
  - Problem determination for the UDDI
  - Problem determination for the Web Services Gateway
- Investigate the following areas for SOAP-related problems:
  - View the JVM logs for the target application server, and run the Log Analyzer on the server's service log.
  - View the error log of the HTTP server to which the SOAP request is sent.
  - View the run time behavior of the SOAP component in more detail, by enabling trace for `org.apache.soap.*` and `com.ibm.*.soap*`.
  - Browse the Web site http://xml.apache.org/soap/ for FAQs and known SOAP issues.

If none of these steps solves the problem, check to see if the problem has been identified and documented using the links in Diagnosing and fixing problems: Resources for learning. If you do not see a problem that resembles yours, or if the information provided does not solve your problem, contact IBM support for further assistance.

Enterprise bean and EJB container troubleshooting tips

If you are having problems starting an EJB container, or encounter error messages or exceptions that appear to be generated on by an EJB container, follow these steps to resolve the problem:

- Browse the relevant log files for clues:
  - Use the Administrative Console to verify that the application server which hosts the container is running.
  - Browse the JVM log files for the application server which hosts the container. Look for the message `server <server_name> open for e-business` in the `SystemOut.log`. If it does not appear, or if you see the message `problems occurred during startup`, browse the `SystemErr.log` for details.
  - Browse the system log files for the application server which hosts the container.
- Use the Log Analyzer tool to browse the service log file for more information.
- Enable tracing for the EJB Container component, by using the following trace specification `EJBContainer=all=enabled`. Follow the instructions for dumping and browsing the trace output to narrow the origin of the problem.

If none of these steps solves the problem, check to see if the problem is identified and documented using the links in Diagnosing and fixing problems: Resources for learning. If you do not see a problem that resembles yours, or if the information provided does not solve your problem, contact IBM support for further assistance.

Security components troubleshooting tips

This document explains basic resources and steps for diagnosing security related issues in the WebSphere Application Server, including:

- What log files to look at and what to look for in them.
- A general approach to isolating and resolving security problems.
- When and how to enable security-related trace.
- An overview and table of security-related CORBA minor codes.

The following security-related problems are addressed elsewhere in this InfoCenter:

- Errors and access problems after enabling security
- Errors after enabling SSL, or SSL-related error messages
- Errors trying to configure and enable security

If none of these steps solves the problem, check to see if the problem has been identified and documented using the links in Diagnosing and fixing problems: Resources for learning. If you do not see a problem that resembles yours, or if the information provided does not solve your problem, contact IBM support for further assistance.

Note: for an overview of WebSphere Application Server security components such as SAS and how they work, see Getting started with security.

Log files

When troubleshooting the security component, browse the JVM logs for the server which is hosting the resource you are trying to access. The following is a sample of messages you would expect to see from a server in which the security service has started successfully:

```
SASRas        A JSAS0001I: Security configuration initialized.
SASRas        A JSAS0002I: Authentication protocol: CSIV2/IBM
SASRas        A JSAS0003I: Authentication mechanism: SWAM
SASRas        A JSAS0004I: Principal name: MYHOSTNAME/aServerID
SASRas        A JSAS0005I: SecurityCurrent registered.
SASRas        A JSAS0006I: Security connection interceptor initialized.
SASRas        A JSAS0007I: Client request interceptor registered.
SASRas        A JSAS0008I: Server request interceptor registered.
SASRas        A JSAS0009I: IOR interceptor registered.
NameServerImp I NMSV0720I: Do Security service listener registration.
SecurityCompo A SECJ0242A: Security service is starting
UserRegistryI A SECJ0136I: Custom Registry:
com.ibm.ws.security.registry.nt.NTLocalDomainRegistryImpl
has been initialized
SecurityCompo A SECJ0202A: Admin application initialized successfully
SecurityCompo A SECJ0203A: Naming application initialized successfully
SecurityCompo A SECJ0204A: Rolebased authorizer initialized successfully
SecurityCompo A SECJ0205A: Security Admin mBean registered successfully
SecurityCompo A SECJ0243A: Security service started successfully
SecurityCompo A SECJ0210A: Security enabled true
```

The following is an example of messages from a server which cannot start the
security service, in this case because the administrative user ID and password
given to communicate with the user registry is wrong, or the user registry itself is
down or misconfigured:

```
SASRas        A JSAS0001I: Security configuration initialized.
SASRas        A JSAS0002I: Authentication protocol: CSIV2/IBM
SASRas        A JSAS0003I: Authentication mechanism: SWAM
SASRas        A JSAS0004I: Principal name: MYHOSTNAME/aServerID
SASRas        A JSAS0005I: SecurityCurrent registered.
SASRas        A JSAS0006I: Security connection interceptor initialized.
SASRas        A JSAS0007I: Client request interceptor registered.
SASRas        A JSAS0008I: Server request interceptor registered.
SASRas        A JSAS0009I: IOR interceptor registered.
NameServerImp I NMSV0720I: Do Security service listener registration.
SecurityCompo A SECJ0242A: Security service is starting
UserRegistryI A SECJ0136I: Custom Registry:
com.ibm.ws.security.registry.nt.NTLocalDomainRegistryImpl has been initialized
Authenticatio E SECJ4001E: Login failed for badID/<null>
javax.security.auth.login.LoginException: authentication failed: bad user/password
```

The following is an example of messages from a server for which LDAP has been
specified as the security mechanism, but the LDAP keys have not been properly
configured:

```
SASRas        A JSAS0001I: Security configuration initialized.
SASRas        A JSAS0002I: Authentication protocol: CSIV2/IBM
SASRas        A JSAS0003I: Authentication mechanism: LTPA
SASRas        A JSAS0004I: Principal name: MYHOSTNAME/anID
SASRas        A JSAS0005I: SecurityCurrent registered.
SASRas        A JSAS0006I: Security connection interceptor initialized.
SASRas        A JSAS0007I: Client request interceptor registered.
SASRas        A JSAS0008I: Server request interceptor registered.
SASRas        A JSAS0009I: IOR interceptor registered.
NameServerImp I NMSV0720I: Do Security service listener registration.
SecurityCompo A SECJ0242A: Security service is starting
UserRegistryI A SECJ0136I: Custom Registry:
com.ibm.ws.security.registry.nt.NTLocalDomainRegistryImpl has
been initialized
SecurityServe E SECJ0237E: One or more vital LTPAServerObject configuration
attributes are null or not available. The attributes and values are password :
LTPA password does exist, expiration time 30, private key <null>,
public key <null>, and shared key <null>.
```

A problem with the SSL configuration might lead to the following message. You
should ensure that the keystore location and keystore passwords are valid. Also,
ensure the keystore has a valid personal certificate and that the personal certificate
public key or CA root has been extracted on put into the truststore.

```
SASRas        A JSAS0001I: Security configuration initialized.
SASRas        A JSAS0002I: Authentication protocol: CSIV2/IBM
SASRas        A JSAS0003I: Authentication mechanism: SWAM
SASRas        A JSAS0004I: Principal name: MYHOSTNAME/aServerId
SASRas        A JSAS0005I: SecurityCurrent registered.
SASRas        A JSAS0006I: Security connection interceptor initialized.
SASRas        A JSAS0007I: Client request interceptor registered.
SASRas        A JSAS0008I: Server request interceptor registered.
SASRas        A JSAS0009I: IOR interceptor registered.
SASRas        E JSAS0026E: [SecurityTaggedComponentAssistorImpl.register]
Exception connecting object to the ORB.  Check the SSL configuration to ensure
that the SSL keyStore and trustStore properties are set properly.  If the problem persists,
contact support for assistance. org.omg.CORBA.OBJ_ADAPTER:
ORB_CONNECT_ERROR (5) - couldn't get Server Subcontract  minor code: 4942FB8F  completed: No
```

General approach for troubleshooting security-related issues

When troubleshooting security-related problems, the following questions are very helpful and should be considered:

Does the problem occur when security is disabled?
> This is a good litmus test to determine that a problem is security related. However, just because a problem only occurs when security is enabled does not always make it a security problem. More troubleshooting is necessary to ensure the problem is really security-related.

Did security appear to initialize properly?
> A lot of security code is visited during initialization. So you will likely see problems there first if the problem is configuration related. The following sequence of messages generated in the SystemOut.log indicate normal code initialization of an application server. This will vary based on the configuration, but the messages are similar:

```
SASRas       A JSAS0001I: Security configuration initialized.
SASRas       A JSAS0002I: Authentication protocol: CSIV2/IBM
SASRas       A JSAS0003I: Authentication mechanism: SWAM
SASRas       A JSAS0004I: Principal name: BIRKT20/pbirk
SASRas       A JSAS0005I: SecurityCurrent registered.
SASRas       A JSAS0006I: Security connection interceptor initialized.
SASRas       A JSAS0007I: Client request interceptor registered.
SASRas       A JSAS0008I: Server request interceptor registered.
SASRas       A JSAS0009I: IOR interceptor registered.
NameServerImp I NMSV0720I: Do Security service listener registration.
SecurityCompo A SECJ0242A: Security service is starting
UserRegistryI A SECJ0136I: Custom Registry:
com.ibm.ws.security.registry.nt.NTLocalDomainRegistryImpl
has been initialized
SecurityCompo A SECJ0202A: Admin application initialized successfully
SecurityCompo A SECJ0203A: Naming application initialized successfully
SecurityCompo A SECJ0204A: Rolebased authorizer initialized successfully
SecurityCompo A SECJ0205A: Security Admin mBean registered successfully
SecurityCompo A SECJ0243A: Security service started successfully
SecurityCompo A SECJ0210A: Security enabled true
```

Is there a stack trace or exception printed in the SystemOut.log?
> A single stack trace tells a lot about the problem. What code initiated the code that failed? What is the failing component? Which class did the failure actually come from? Sometimes the stack trace is all that is needed to solve the problem and it can pinpoint the root cause. Other times, it can only give us a clue, and could actually be misleading. When support analyzes a stack trace, they may request additional trace if it is not clear what the problem is. If it appears to be security related and the solution cannot be determined from the stack trace or problem description, you will be asked to gather the following trace specification:
> SASRas=all=enabled:com.ibm.ws.security.*=all=enabled from all processes involved.

Is this a distributed security problem or a local security problem?
> - If the problem is local, that is the code involved does not make a remote method invocation, then troubleshooting is isolated to a single process. It is important to know when a problem is local versus distributed since the behavior of the Orb, among other components, is different between the two. Once a remote method invocation takes place, an entirely different security code path is entered.
> - When you know that the problem involves two or more servers, the techniques of troubleshooting change. You will need to trace all servers involved simultaneously so that the trace shows the client and server sides of the problem. Try to make sure the timestamps on all machines

match as closely as possible so that you can find the request and reply pair from two different processes. Enable both SAS and Security trace using the trace specification:
`SASRas=all=enabled:com.ibm.ws.security.*=all=enabled`.

Is the problem related to authentication or authorization?

Most security problems fall under one of these two categories. Authentication is the process of determing who the caller is. Authorization is the process of validating that the caller has the proper authority to invoke the requested method. When authentication fails, typically this is related to either the authentication protocol, authentication mechanism or user registry. When authorization fails, this is usually related to the application bindings from assembly and/or deployment and to the caller's identity who is accessing the method and the roles required by the method.

Is this a Web or EJB request?

Web requests have a completely different code path than EJB requests. Also, there are different security features for Web requests than for EJB requests, requiring a completely different body of knowledge to resolve. For example, when using the LTPA authentication mechanism, the Single SignOn feature is available for Web requests but not for EJB requests. Web requests involve HTTP header information not required by EJB requests due to the protocol differences. Also, the Web container (or servlet engine) is involved in the entire process. Any of these components could be involved in the problem and all should be considered during troubleshooting, based on the type of request and where the failure occurs.

Secure EJB requests heavily involve the ORB and Naming components since they flow over the RMI/IIOP protocol. In addition, when work flow management (WLM) is enabled, other behavior changes in the code can be observed. All of these components interact closely for security to work properly in this environment. At times, trace in any or all of these components might be necessary to troubleshoot problems in this area. The trace specification to begin with is
`SASRas=all=enabled:com.ibm.ws.security.*=all=enabled`. ORB trace is also very beneficial when the SAS/Security trace does not seem to pinpoint the problem.

Does the problem seem to be related to SSL?

The Secure Socket Layer is just that, a totally distinct, separate layer of security. Troubleshooting SSL problems are usually separate from troubleshooting authentication and/or authorization problems. There are many things to consider. Usually, SSL problems are first time setup problems because the configuration can be difficult. Each client must contain the server's signer certificate. During mutual authentication, each server must contain the client's signer certificate. Also, there can be protocol differences (SSLv3 vs. TLS), and listener port problems related to stale IORs (i.e., IORs from a server reflecting the port prior to the server restarting).

For SSL problems, we sometimes request an SSL trace to determine what is happening with the SSL handshake. The SSL handshake is the process which occurs when a client opens a socket to a server. If anything goes wrong with the key exchange, cipher exchange, etc. the handshake will fail and thus the socket is invalid. Tracing JSSE (the SSL implementation used in WebSphere Application Server) involves the following steps:

- Ensure that the client and server processes contain an `ibmjsse-debug.jar` file in the `java/jre/lib/ext` directory. The `ibmjsse-debug.jar` is shipped with the product. You can locate the file under `<installation_directory>\web\docs\jsse`. Make sure you remove the existing `ibmjsse.jar` file from this directory after putting in the `ibmjsse-debug.jar`. If both exist in the `/ext` directory, the JSSE trace will not be complete.
- Set the following system property on the client and server processes: `-Djavax.net.debug=true`. For the server, add this to the Generic JVM Arguments property of the Java virtual machine settings page.
- Turn on ORB trace as well.
- Recreate the problem. The `SystemOut.log` of both processes should contain the JSSE trace. You will find trace similar to the following:

```
SSLConnection: install <com.ibm.sslite.e@3ae78375>
>> handleHandshakeV2 <com.ibm.sslite.e@3ae78375>
>> handshakeV2 type = 1
>> clientHello: SSLv2.
SSL client version: 3.0
...
...
...
JSSEContext: handleSession[Socket[addr=null,port=0,localport=0]]

<< sendServerHello.
SSL version: 3.0
SSL_RSA_WITH_RC4_128_MD5
HelloRandom
...
...
...
<< sendCertificate.
<< sendServerHelloDone.
>> handleData <com.ibm.sslite.e@3ae78375>
>> handleHandshake <com.ibm.sslite.e@3ae78375>
>> handshakeV3 type = 16
>> clientKeyExchange.
>> handleData <com.ibm.sslite.e@3ae78375>
>> handleChangeCipherSpec <com.ibm.sslite.e@3ae78375>
>> handleData <com.ibm.sslite.e@3ae78375>
>> handleHandshake <com.ibm.sslite.e@3ae78375>
>> handshakeV3 type = 20
>> finished.
<< sendChangeCipherSpec.
<< sendFinished.
```

Tracing security

The classes which implement WebSphere Application Server security are:

- com.ibm.ws.security.*
- com.ibm.websphere.security.*
- com.ibm.WebSphereSecurityImpl.*
- SASRas

To view detailed information on the runtime behavior of security, enable trace on the following components and review the output:

- `com.ibm.ws.security.*=all=enabled:com.ibm:WebSphereSecurityImpl.*=all=enabled:com.ibm.websphere.security.*=all=enabled`. This trace statement collects the trace for the security runtime.

- `com.ibm.ws.console.security.*=all=enabled`. This trace statement collects the trace for the security center GUI.
- `SASRas=all=enabled`. This trace statement collects the trace for SAS (low-level authentication logic).

Fine tuning SAS traces:
> If a subset of classes need to be traced for the SAS/CSIv2 component, a system property can be specified with the class names comma separated: `com.ibm.CORBA.securityTraceFilter=SecurityConnectionInterceptorImpl, VaultImpl, ...`

Fine tuning Security traces:
> If a subset of packages need to be traced, specify a trace specification more detailed than `com.ibm.ws.security.*=all=enabled`. For example, to trace just dynamic policy code, you can specify `com.ibm.ws.security.policy.*=all=enabled`. To disable dynamic policy trace, you can specify `com.ibm.ws.security.policy.*=all=disabled`.

Configuring CSIv2 or SAS Trace Settings
> Situations arise where reviewing trace for the CSIv2 or SAS authentication protocols can assist in troubleshooting difficult problems. This section decribes how to enable to CSIv2/SAS trace.

Enabling Client-Side CSIv2/SAS Trace
> To enable CSIv2 and SAS trace on a pure client, the following steps need to be taken:
> - Edit the file TraceSettings.properties in the **/WebSphere/AppServer/properties directory**.
> - In this file, change `traceFileName=` to point to the path in which you want the ouput file created. Make sure you put a double backslash (\\) between each subdirectory. For example, `traceFileName=c:\\WebSphere\\AppServer\\logs\\sas_client.log`
> - In this file, add the trace specification string: `SASRas=all=enabled`. Any additional trace strings can be added on separate lines.
> - Point to this file from within your client application. On the Java command line where you launch the client, add the following system property: `-DtraceSettingsFile=TraceSettings.properties`.
>
>   **Note:** Do not give the fully qualified path to the `TraceSettings.properties` file. Make sure that the `TraceSettings.properties` file is in your classpath.

Enabling Server-Side CSIv2/SAS Trace
> To enable SAS trace in an application server, complete the following:
> - Add the trace specification, `SASRas=all=enabled`, to the `server.xml` file or add it to the Trace settings within the WebConsole GUI.
> - Typically it is best to also trace the authorization security runtime in addition to the authentication protocol runtime. To do this, use the following two trace specifications in combination: `SASRas=all=enabled:com.ibm.ws.security.*=all=enabled`.
> - When troubleshooting a connection type problem, it is beneficial to trace both SAS/CSIv2 and the ORB. To do this, use the following three trace specifications:

SASRas=all=enabled:com.ibm.ws.security.*=all=enabled:
ORBRas=all=enabled

.

- In addition to adding these trace specifications, for ORB trace there are a couple of system properties that also need to be set. Go to the ORB settings in the GUI and add the following two properties: `com.ibm.CORBA.Debug=true` and `com.ibm.CORBA.CommTrace=true`.

CSIv2 CORBA Minor Codes

Whatever exceptions might occur within the security code on either the client or server, the eventual exception will become a CORBA exception. So any exception that occurs gets "wrapped" by a CORBA exception, because the CORBA architecture is used by the security service for its own inter-process communication. CORBA exceptions are generic, and indicate a problem in communication between two components. CORBA minor codes are more specific, and indicate the underlying reason that a component could not complete a request.

The following shows the CORBA Minor codes which a client can expect to receive after executing a security-related request such as authentication. It also includes the CORBA exception type that the minor code would appear in.

The following exception shows an example of a CORBA exception where the minor code is 49424300. From the table below, this minor code indicates Authentication Failure. Typically, a descriptive message is also included in the exception to assist in troubleshooting the problem. Here, the detailed message is "Exception caught invoking authenticateBasicAuthData from SecurityServer for user jdoe. Reason: com.ibm.WebSphereSecurity.AuthenticationFailedException" which indicates that the authentication failed for user "jdoe".

The completed field in the exception indicates whether the method was completed or not. In the case of a `NO_PERMISSION`, the method should never get invoked, so it will always be "completed:No". Other exceptions which are caught on the server side could have a completed status of "Maybe" or "Yes".

```
org.omg.CORBA.NO_PERMISSION: Caught WSSecurityContextException in
WSSecurityContext.acceptSecContext(),
reason: Major Code[0] Minor Code[0] Message[Exception caught invoking
authenticateBasicAuthData from SecurityServer for user jdoe.
Reason: com.ibm.WebSphereSecurity.AuthenticationFailedException]
minor code: 49424300  completed: No

at com.ibm.ISecurityLocalObjectBaseL13Impl.PrincipalAuthFailReason.map_auth_
fail_to_minor_code(PrincipalAuthFailReason.java:83)
at com.ibm.ISecurityLocalObjectBaseL13Impl.CSIServerRI.receive_request(CSIServerRI.java:1569)
at com.ibm.rmi.pi.InterceptorManager.iterateReceiveRequest(InterceptorManager.java:739)
at com.ibm.CORBA.iiop.ServerDelegate.dispatch(ServerDelegate.java:398)
at com.ibm.rmi.iiop.ORB.process(ORB.java:313)
at com.ibm.CORBA.iiop.ORB.process(ORB.java:1581)
at com.ibm.rmi.iiop.GIOPConnection.doWork(GIOPConnection.java:1827)
at com.ibm.rmi.iiop.WorkUnitImpl.doWork(WorkUnitImpl.java:81)
at com.ibm.ejs.oa.pool.PooledThread.run(ThreadPool.java:91)
at com.ibm.ws.util.CachedThread.run(ThreadPool.java:149)
```

The following table shows the CORBA Minor codes which a client can expect to receive after executing a security-related request such as authentication. It also includes the CORBA exception type that the minor code would appear in.

Minor code name	Minor code value (in hex)	Exception type (all in the package of org.omg.CORBA .*)	Minor code description	Retry performed (when authenticationRe tryEnabled=true)
AuthenticationFailed	49424300	NO_PERMISSION	This is a generic authentication failed error. It does not give any details about whether the userid or password is invalid. Some registries can choose to use this type of error code, others might choose to use the next three types which are more specific.	Yes
InvalidUserid	49424301	NO_PERMISSION	This occurs when the registry returns bad userid.	Yes
InvalidPassword	49424302	NO_PERMISSION	This occurs when the registry returns bad password.	Yes
InvalidSecurityCredentials	49424303	NO_PERMISSION	This is a generic error indicating that the credentials are bad for whatever reason. It could be that they don't have the right attributes set.	Yes, if client has BasicAuth credential (token based credential was rejected in the first place).
InvalidRealm	49424304	NO_PERMISSION	This occurs when the REALM in the token received from the client does not match the server's current realm.	No
ValidationFailed	49424305	NO_PERMISSION	A validation failure occurs when a token is sent from the client or server to a target server but the token format or the expiration is invalid.	Yes, if client has BasicAuth credential (token based credential was rejected in the first place).

CredentialTokenExpired	49424306	NO_PERMISSION	This is more specific about why the validation failed. In this case, the token has a absolute lifetime, and this lifetime has expired. Therefore, it is no longer a valid token and cannot be used.	Yes, if client has BasicAuth credential (token based credential was rejected in the first place).
InvalidCredentialToken	49424307	NO_PERMISSION	This is more specific about why the validation failed. In this case, the token cannot be decrypted or the data within it is not readable.	Yes, if client has BasicAuth credential (token based credential was rejected in the first place).
SessionDoesNotExist	49424308	NO_PERMISSION	This indicates that the CSIv2 session does not exist on the server. Typically, a retry occurs automatically and will successfully create a new session.	Yes
SessionConflictingEvidence	49424309	NO_PERMISSION	This indicates that a session already exists on the server which matches the context_id sent over by the client, however, the information provided by the client for this EstablishContext message is different from the information originally provided to establish the session.	Yes
SessionRejected	4942430A	NO_PERMISSION	This indicates that the session referenced by the client has been previously rejected by the server.	Yes

SecurityServerNotAvailable	4942430B	NO_PERMISSION	This error occurs when the server cannot contact the security server (whether local or remote) in order to authenticate or validate.	No
InvalidIdentityToken	4942430C	NO_PERMISSION	This error indicates that identity cannot be obtained from the identity token when Identity Assertion is enabled.	No
IdentityServerNotTrusted	4942430D	NO_PERMISSION	This indicates that the server id of the sending server is not on the target server's trusted principal list.	No
InvalidMessage	4942430E	NO_PERMISSION	This indicates that the CSIv2 message format is invalid for the receiving server.	No
AuthenticationNotSupported	49421090	NO_PERMISSION	This error occurs when a mechanism does not support authentication (very rare).	No
InvalidSecurityMechanism	49421091	NO_PERMISSION	This is used to indicate that the specified security mechanism is not known.	No
CredentialNotAvailable	49421092	NO_PERMISSION	This indicates a credential is not available when it is required.	No
SecurityMechanismNotSupported	49421093	NO_PERMISSION	This error occurs when a security mechanism specified in the CSIv2 token is not implemented on the server.	No

ValidationNotSupported	49421094	NO_PERMISSION	This error occurs when a mechanism does not support validation (such as LocalOS). This error should not occur since the LocalOS credential is not a forwardable credential, therefore, validation should never need to be called on it.	No
CredentialTokenNotSet	49421095	NO_PERMISSION	This is used to indicate the token inside the credential is null.	No
ServerConnectionFailed	494210A0	COMM_FAILURE	This error is used when a connection attempt fails.	Yes (via Orb retry)
CorbaSystemException	494210B0	INTERNAL	This is a generic Corba specific exception in system code.	No
JavaException	494210B1	INTERNAL	This is a generic error that indicated an unexpected Java exception occurred.	No
ValueIsNull	494210B2	INTERNAL	This is used to indicate that a value or parameter passed in was null.	No
EffectivePolicyNotPresent	494210B3	INTERNAL	This indicates that an effective policy object for CSIv2 is not present. This object is used to determine what security configuration features have been specified.	No
NullPointerException	494210B4	INTERNAL	This is used to indicate that a NullPointerException was caught in the runtime.	No
ErrorGettingClassInstance	494210B5	INTERNAL	This indicates a problem loading a class dynamically.	No
MalFormedParameters	494210B6	INTERNAL	This indicates parameters are not valid.	No

DuplicateSecurityAttributeType	494210B7	INTERNAL	A duplicate credential attribute has been specified during the set_attributes operation.	No
MethodNotImplemented	494210C0	NO_IMPLEMENT	A method invoked has not been implemented.	No
GSSFormatError	494210C5	BAD_PARAM	This indicates that a GSS encoding or decoding routine has thrown an exception.	No
TagComponentFormatError	494210C6	BAD_PARAM	This indicates that a tag component cannot be read properly.	No
InvalidSecurityAttributeType	494210C7	BAD_PARAM	This indicates an attribute type specified during the set_attributes operation is an invalid type.	No
SecurityConfigError	494210CA	INITIALIZE	A problem exists between the client and server configuration.	No

JSP engine troubleshooting tips

If you are having difficulty using the JSP engine, try these steps:

1. Determine whether other resources such as `.html` files or servlets are being requested and displayed correctly. If they are not, the problem probably lies at a deeper level, such as with the HTTP server.

2. If other resources are being displayed correctly, determine whether the JSP engine has started normally:

   - Browse the JVM logs of the server hosting the JSP files you are trying to access. A message such as `<application_name/Sevlet.LOG>: JSP 1.2 Processor: init"` in the `<root_dir>/logs/<server_name>/SystemOut.log` file indicates that the JSP engine has started normally. If the JSP processor fails to load, you may see a message such as `<strong>Did not realize init() exception thrown by servlet JSP 1.2 Processor</strong>` in `<application_name/Servlet.LOG>: JSP 1.2 Processor: init"` in the `<root_dir>/logs/<server_name>/SystemOut.log` file.

   - Open the Log analyzer on the service log of the server which is hosting the jsp you are trying to access and use it to browse error and warning messages.

3. If the JSP engine has started normally, the problem may be with the JSP file itself.

   - Copy a simple JSP file (such as the WebSphere Application Server sample "HelloHTML.jsp") to the Web application's document root and attempt to serve it.

   - If that works, examine the target application server's `SystemOut.log` for invalid JSP directive syntax . Errors similar to the following in a browser

indicate this kind of problem: Message: /<jspname>.jsp(9,0) Include: Mandatory attribute page missing. This example indicates that line 9, column 0 of the named JSP is missing a mandatory page attribute. Similar messages are displayed for other syntax errors.

- Examine the target application server's `SystemErr.log` files for problems with invalid Java syntax. Errors similar to Message: Unable to compile class for JSP in a browser indicate this kind of problem.

  The error message output from the Javac compiler will be found in the `SystemErr.log`. It might look like:

  ```
  C:\WASROOT\temp\ ... test.war\_myJsp.java:14: Duplicate variable declaration:
  int myInt was int myInt int myInt = 122;
  String myString = "number is 122"; static int myStaticInt=22; int myInt=121;
  1 error
  ```

  Correct the error in the JSP file and retry the file.

If none of these steps solves the problem, check to see if the problem is identified and documented using the links in Diagnosing and fixing problems: Resources for learning. If you do not see a problem that resembles yours, or if the information provided does not solve your problem, contact IBM support for further assistance.

Workload Management component troubleshooting tips

If the Workload Management component is not properly distributing the workload across servers in multi-node configuration, use these steps to isolate the problem.

There are four basic steps troubleshooting the Workload Management component:
- Eliminate environment or configuration issues
- Browse log files for WLM errors and WLM CORBA minor codes
- Analyze PMI data
- Resolve problem or contact IBM support

Eliminate environment or configuration issues

First, determine the health of the cluster. In other words, are the servers capable of serving the applications for which they have been enabled? To do this, you must identify the cluster that is exhibiting the problem.
- Are there network connection problems with the members of the cluster or the administrative servers, for example deployment manager or node agents?
  - If so, `ping` the machines to ensure that they are properly connected to the network.
- Is there other activity on the machines where the servers are installed that is impacting the servers ability to service a request? For example, check the processor utilization as measured by the task manager, processor ID, or some other outside tool to see if:
  - It is not what is expected, or is erratic rather than constant.
  - It shows that a newly added, installed, or upgraded member of the cluster is not being utilized.
- Are all of the application servers you started on each node running, or are some stopped?
- Are the applications installed and operating?
- If the problem relates to distributing workload across persistent (CMP or BMP) enterprise beans, have you configured the supporting JDBC drivers and datasources on each server? For problems relating to data access, review the topic Cannot access a datasource.

If you are experiencing workload management problems related to HTTP requests, such as HTTP requests not being served by all members of the cluster, be aware that the HTTP plugin will balance the load across all servers that are defined in the PrimaryServers list if affinity has not been established. If you do not have a PrimaryServers list defined then the plugin will load balance across all servers defined in the cluster if affinity has not been established. If affinity has been established, the plugin should go directly to that server for all requests.

For workload management problems relating to enterprise bean requests, such as enterprise bean requests not getting served by all members of a cluster:

- Are the weights set to the allowed values?
  - For the cluster in question, log onto the administrative console and:
    1. Select **Cluster -> Manage cluster**
    2. Select your cluster from the list.
    3. Select **Cluster Members**.
    4. For each server in the cluster, click on *servername* and note the assigned weight of the server.
  - Ensure that the weights are within the valid range of 0-20. If a server has a weight of 0, no requests will be routed to it. Weights greater than 20 are treated as 0.

Note: The remainder of this article deals with enterprise bean workload balancing only. For more help on diagnosing problems in distributing Web (HTTP) requests, view the topics HTTP plugin component troubleshooting tips and Web resource (JSP, servlet, html file, image, etc) will not display.

Browse log files for WLM errors and CORBA minor codes

If you still encounter problems with enterprise bean workload management, the next step is to check the activity log for entries that show:

- A server that has been marked unusable more than once and remains unusable.
- All servers in a cluster have been marked bad and remain unusable.
- A Location Service Daemon (LSD) has been marked unusable more than once and remains unusable.

To do this, use the Log Analyzer tool to open the service log (`activity.log`) on the affected servers, and look for the following entries:

- **WWLM0061W**: An error was encountered sending a request to cluster member *member* and that member has been marked unusable for future requests to the cluster *cluster*.

  Note: It is not unusual for a server to be marked unusable. The server may be tagged unusable for normal operational reasons, such as a ripple start being executed while there is still a load on the server from a client.

- **WWLM0062W**: An error was encountered sending a request to cluster member *member* that member has been marked unusable, for future requests to the cluster *cluster* two or more times.
- **WWLM0063W**: An error was encountered attempting to use the LSD *LSD_name* to resolve an object reference for the cluster *cluster* and has been marked unusable for future requests to that cluster.
- **WWLM0064W**: Errors have been encountered attempting to send a request to all members in the cluster *cluster* and all of the members have been marked unusable for future requests that cluster.

- WWLM0065W: An error was encountered attempting to update a cluster member *server* in cluster *cluster*, as it was not reachable from the deployment manager.

If any of these warning are encountered, follow the user response given in the log. If, after following the user response, the warnings persist, look at any other errors and warnings in the Log Analyzer on the affected servers to look for:

- A possible user response, such as changing a configuration setting.
- Base class exceptions that might indicate a WebSphere Application Server defect.

You may also see exceptions with "CORBA" as part of the exception name, since WLM uses CORBA (Common Object Request Broker Architecture) to communicate between processes. Look for a statement in the exception stack specifying a "minor code". These codes denote the specific reason a CORBA call or response could not complete. WLM minor codes fall in range of 0x4921040 - 0x492104F. For an explanation of minor codes related to WLM, see the Javadocfor the package and class com.ibm.websphere.wlm.WsCorbaMinorCodes.

Analyze PMI data

The purpose for analyzing the PMI data is to understand the workload arriving for each member of a cluster. The data for any one member of the cluster is only useful within the context of the data of all the members of the cluster. To obtain PMI data for all members of a cluster, see (Performance monitoring infrastructure).

Once you have obtained the PMI data, you should calculate the percentage of numIncomingRequests for each member of the cluster to the total of the numIncomingRequests of all members of the cluster. A comparison of this percentage value to the percentage of weights directed to each member of the cluster provides an initial look at the balance of the workload directed to each member of a cluster.

In addition to the numIncomingRequests two other metrics show how work is balanced between the members of a cluster, numincomingStrongAffinityRequests and numIncomingNonWLMObjectRequests. These two metrics show the number of requests directed to a specific member of a cluster that could only be serviced by that member.

For example, consider a 3-server cluster. We have assigned the following weights to each of these three servers:

- Server1 = 5
- Server2 = 3
- Server3 = 2

Allow our cluster of servers to start servicing requests, and wait for the system to reach a steady state, that is the number of incoming requests to the cluster equals the number of responses from the servers. In such a situation, we would expect that the percentage of requests routed to each server to be:

- % routed to Server1 = weight1 / (weight1+weight2+weight3) = 5/10 or 50%
- % routed to Server2 = weight2 / (weight1+weight2+weight3) = 3/10 or 30%
- % routed to Server3 = weight3 / (weight1+weight2+weight3) = 2/10 or 20%

Now let us consider a case where there are no incoming requests with neither strong affinity nor any non-WLM object requests.

In this scenario, let us assume that the PMI metrics gathered show the number of incoming requests for each server are:

- numIncomingRequestsServer1 = 390
- numIncomingRequestsServer2 = 237
- numIncomingRequestsServer3 = 157

Thus, the total number of requests coming into the cluster is: numIncomingRequestsCluster = numIncomingRequestsServer1 + numIncomingRequestsServer2 + numIncomingRequestsServer3 = 784

numincomingStrongAffinityRequests = 0

numIncomingNonWLMObjectRequests = 0

Can we decide based on this data if WLM is properly balancing the incoming requests among the servers in our cluster? Since there are no requests with strong affinity, the question we need to answer is, are the requests in the ratios we expect based on the assigned weights? The computation to answer that question is straightforward:

- % (actual) routed to Server1 = 390 / 784 = 49.8%
- % (actual) routed to Server2 = 237 / 784 = 30.2%
- % (actual) routed to Server3 = 157 / 784 = 20.0%

So WLM is behaving as designed, as the data are completely what is expected, based on the weights assigned the servers.

Now let us consider a 3-server cluster. We have assigned the following weights to each of these three servers:

- Server1 = 5
- Server2 = 3
- Server3 = 2

Allow this cluster of servers to start servicing requests and wait for the system to reach a steady state, that is the number of incoming requests to the cluster equals the number of responses from the servers. In such a situation, we would expect that the percentage of requests routed to Server1-3 would be:

- % routed to Server1 = weight1 / (weight1+weight2+weight3) = 5/15 or 1/3 of the requests.
- % routed to Server2 = weight2 / (weight1+weight2+weight3) = 5/15 or 1/3 of the requests.
- % routed to Server3 = weight3 / (weight1+weight2+weight3) = 5/15 or 1/3 of the requests.

In this scenario, let us assume that the PMI metrics gathered show the number of incoming requests for each server are:

- numIncomingRequestsServer1 = 1236
- numIncomingRequestsServer2 = 1225
- numIncomingRequestsServer3 = 1230

Thus, the total number of requests coming into the cluster:

- numIncomingRequestsCluster = numIncomingRequestsServer1 + numIncomingRequestsServer2 + numIncomingRequestsServer3 = 3691

- numincomingStrongAffinityRequests = 445, and that all 445 requests are aimed at Server1.
- numIncomingNonWLMObjectRequests = 0.

In this case, we see that the number of requests was not evenly split among the three servers, as expected. Instead, the distribution is:
- % (actual) routed to Server1 = 1236 / 3691= 33.49%
- % (actual) routed to Server2 = 1225 / 3691= 33.19%
- % (actual) routed to Server3 = 1230 / 3691= 33.32%

However, the correct interpretation of this data is the routing of requests is not perfectly balanced because Server1 had several hundred strong affinity requests. WLM attempts to compensate for strong affinity requests directed to 1 or more servers by distributing new incoming requests preferentially to servers which are not participating in transactional affinity, to compensate for those servers that are participating in transactions. In the case of incoming requests with strong affinity and non-WLM object requests, the analysis would be analogous to this case.

If, once you have analyzed the PMI data and accounted for transactional affinity and non-WLM object requests, the percentage of actual incoming requests to servers in a cluster to do not reflect the assigned weights, this indicates that requests are not being properly balanced. If this is the case, it is recommended that you repeat the steps described above for eliminating environment and configuration issues and browsing log files before proceeding.

Resolve problem or contact IBM support
If the PMI data or client logs indicate an error in WLM, collect the following information and contact IBM support.
- A detailed description of your environment.
- A description of the symptoms.
- The SystemOut.logs and SystemErr.logs for all servers in the cluster.
- The activity.log.
- The First Failure Data Capture logs.
- The PMI metrics.
- A description of what the client is attempting to do, and a description of the client. For example, 1 thread, multiple threads, servlet, J2EE client, etc..

If none of these steps solves the problem, check to see if the problem has been identified and documented using the links in Diagnosing and fixing problems: Resources for learning. If you do not see a problem that resembles yours, or if the information provided does not solve your problem, contact IBM support for further assistance.

Object Request Broker component troubleshooting tips

This article describes how to diagnose problems related to the WebSphere Application Server Object Request Broker (ORB) component by explaining:
- How to enable tracing for the ORB component.
- What log files to examine for more information.
- Information on the Java packages containing the ORB Service.
- ORB-related tools.
- Where to find configurable settings.

- A listing of CORBA minor codes generated by this component.

Enabling tracing for the Object Request Broker component

The Object Request Broker (ORB) Service is one of the WebSphere Application Server run time services. Tracing of messages sent and received by the ORB is a useful starting point for troubleshooting the ORB Service. You can selectively enable or disable tracing of ORB messages for each server in a WebSphere Application Server installation, and for each application client.

This tracing is referred to by WebSphere Application Server support as a *comm trace*, and is different from the general purpose trace facility. The trace facility, which shows the detailed run time behavior of product components, may be used alongside comm trace for other product components, or for the ORB component. The trace string associated with the ORB Service is "ORBRas=all=enabled".

You can enable and disable comm tracing using the administrative console or by manually editing the **server.xml** file for the server be trace. You must stop and restart the server for the configuration change to take effect.

For example, using the administrative console:

- Navigate to the desired server by clicking **Servers > Manage Application Servers > server1 > ORB Service**, and select the ORB tracing checkbox. or
- Locate the server.xml file for the selected server, for example:

  <install_dir>/config/cells/<nodename>/nodes/<nodename>/servers/ <servername>/server.xml
- Locate the services entry for the ORB Service (xmi:type="orb:ObjectRequestBroker") and set **commTraceEnabled**="**true**".

To enable ORB comm tracing for client applications, you must specify two ORB properties in the command line used to launch the client application:

- If you are using the WebSphere Application Server launcher launchClient, use the option **-CCD** or
- If you are using the **java** command directly, use the -D option to specify these parameters:
  - com.ibm.CORBA.Debug=true
  - com.ibm.CORBA.CommTrace=true

Log files and messages associated with Object Request Broker

Messages and trace information for the ORB are captured primarily in two logs:

- <install_dir>/logs/<servername>/trace.log for output from comm tracing and tracing the behavior of the ORBRas component, and
- The JVM logs for each application server, for WebSphere Application Server error and warning messages.

In general, if the following message appears in the SystemOut.log file, it indicates the application server has successfully started, which indicates that the ORB Service was successfully started also:

WSVR0001I: Server server1 open for e-business

When comm tracing is enabled, a message in the file *install_dir*/logs/*servername*/trace.log similar to the one below indicates that the

ORB service has started successfully. It also shows a ListenerThread has started successfully and is waiting for requests on the specified local port.

com.ibm.ws.orbimpl.transport.WSTransport startListening(ServerConnectionData connectionData) P=693799:O=0:CT a new ListenerThread has been started for ServerSocket[addr=0.0.0.0/0.0.0.0,port=0,localport=1360]

If tracing of the Object Adapter has been enabled (`com.ibm.ejs.oa.*=all=enabled`), the following message in the trace.log indicates that the ORB service has started successfully:

EJSORBImpl < initializeORB

The ORB service is one of the first services started during the WebSphere Application Server initialization process. If it is not properly configured, other components such as Naming, Security, and Node Agent, are not likely to start successfully. This is obvious in the JVM logs or `trace.log` of the affected application server.

Java packages containing the Object Request Broker service

The ORB service resides in the following Java packages:

- com.ibm.com.CORBA.*
- com.ibm.rmi.*
- com.ibm.ws.orb.*
- com.ibm.ws.orbimpl.*
- org.omg.CORBA.*
- javax.rmi.CORBA.*

The `.jar` files which contain the packages above are:

- `<install_dir>/java/jre/lib/ext/ibmorb.jar`
- `<install_dir>/java/jre/lib/ext/iwsorbutil.jar`
- `<install_dir>/lib/iwsorb.jar`

Tools used with Object Request Broker

The tools used to compile Java remote interfaces to generate language bindings used by the ORB at runtime reside in the following Java packages:

- com.ibm.tools.rmic.*
- com.ibm.idl.*

The `.jar` file which contains the packages is `<install_dir>/java/lib/ibmtools.jar`.

Object Request Broker properties

The ORB service requires a number of ORB properties for correct operation. It is not necessary for most users to modify these properties, and it is recommended that only your system administrator modify them when required.. Consult IBM Support personnel for assistance. The properties reside in the orb.properties file, located at `<install_dir>/java/jre/lib/orb.properties`.

CORBA minor codes

The CORBA specification defines standard minor exception codes for use by the ORB when a system exception is thrown. In addition, the Object Management Group (OMG) assigns each vendor a unique prefix value for use in vendor-proprietary minor exception codes. The minor code values assigned to IBM

and used by the ORB in the WebSphere Application Server are shown below. The minor code value is shown in decimal and hexadecimal formats. The column labeled "Minor Code Reason" gives a short description of the condition causing the exception. Currently there is no documentation for these errors beyond the "Minor Code Reason" shown below. If technical support from IBM is required, the code helps support engineers in determining the source of the problem.

Decimal	Hexadecimal	Minor Code Reason
1229066320	0x49421050	HTTPOUTPUTSTREAM_WRITE
1229066321	0x49421051	COULD_NOT_INSTANTIATE_CLIENT_SSL_SOCKET_FACTORY
1229066322	0x49421052	COULD_NOT_INSTANTIATE_SERVER_SSL_SOCKET_FACTORY
1229066323	0x49421053	CREATE_LISTENER_FAILED_1
1229066324	0x49421054	CREATE_LISTENER_FAILED_2
1229066325	0x49421055	CREATE_LISTENER_FAILED_3
1229066326	0x49421056	CREATE_LISTENER_FAILED_4
1229066327	0x49421057	CREATE_LISTENER_FAILED_5
1229066328	0x49421058	INVALID_CONNECTION_TYPE
1229066329	0x49421059	HTTPINPUTSTREAM_NO_ACTIVEINPUTSTREAM
1229066330	0x4942105a	HTTPOUTPUTSTREAM_NO_OUTPUTSTREAM
1229066331	0x4942105b	CONNECTIONINTERCEPTOR_INVALID_CLASSNAME
1229066332	0x4942105c	NO_CONNECTIONDATA_IN_CONNECTIONDATACARRIER
1229066333	0x4942105d	CLIENT_CONNECTIONDATA_IS_INVALID_TYPE
1229066334	0x4942105e	SERVER_CONNECTIONDATA_IS_INVALID_TYPE
1229066335	0x4942105f	NO_OVERLAP_OF_ENABLED_AND_DESIRED_CIPHER_SUITES
1229066352	0x49421070	CONNECT_FAILURE_ON_SSL_CLIENT_SOCKET
1229066353	0x49421071	GETCONNECTION_KEY_RETURNED_FALSE
1229066354	0x49421072	UNABLE_TO_CREATE_SSL_SOCKET
1229066355	0x49421073	SSLSERVERSOCKET_TARGET_SUPPORTS_LESS_THAN_1
1229066356	0x49421074	SSLSERVERSOCKET_TARGET_REQUIRES_LESS_THAN_1
1229066357	0x49421075	SSLSERVERSOCKET_TARGET_LESS_THAN_TARGET_REQUIRES
1229066358	0x49421076	UNABLE_TO_CREATE_SSL_SERVER_SOCKET
1229066359	0x49421077	CAUGHT_EXCEPTION_WHILE_CONFIGURING_SSL_SERVER_SOCKET
1229066360	0x49421078	INVALID_SERVER_CONNECTION_DATA_TYPE
1229066361	0x49421079	GETSERVERCONNECTIONDATA_RETURNED_NULL
1229066362	0x4942107a	GET_SSL_SESSION_RETURNED_NULL
1229066363	0x4942107b	GLOBAL_ORB_EXISTS
1229123841	0x4942f101	DSIMETHOD_NOTCALLED
1229123842	0x4942f102	BAD_INV_PARAMS
1229123843	0x4942f103	BAD_INV_RESULT
1229123844	0x4942f104	BAD_INV_CTX
1229123879	0x4942f127	PI_NOT_POST_INIT
1229123969	0x4942f181	BAD_OPERATION_EXTRACT_SHORT
1229123970	0x4942f182	BAD_OPERATION_EXTRACT_LONG
1229123971	0x4942f183	BAD_OPERATION_EXTRACT_USHORT
1229123972	0x4942f184	BAD_OPERATION_EXTRACT_ULONG
1229123973	0x4942f185	BAD_OPERATION_EXTRACT_FLOAT
1229123974	0x4942f186	BAD_OPERATION_EXTRACT_DOUBLE
1229123975	0x4942f187	BAD_OPERATION_EXTRACT_LONGLONG
1229123976	0x4942f188	BAD_OPERATION_EXTRACT_ULONGLONG
1229123977	0x4942f189	BAD_OPERATION_EXTRACT_BOOLEAN
1229123978	0x4942f18a	BAD_OPERATION_EXTRACT_CHAR
1229123979	0x4942f18b	BAD_OPERATION_EXTRACT_OCTET
1229123980	0x4942f18c	BAD_OPERATION_EXTRACT_WCHAR
1229123981	0x4942f18d	BAD_OPERATION_EXTRACT_STRING
1229123982	0x4942f18e	BAD_OPERATION_EXTRACT_WSTRING
1229123983	0x4942f18f	BAD_OPERATION_EXTRACT_ANY

1229123984	0x4942f190	BAD_OPERATION_INSERT_OBJECT_1
1229123985	0x4942f191	BAD_OPERATION_INSERT_OBJECT_2
1229123986	0x4942f192	BAD_OPERATION_EXTRACT_OBJECT_1
1229123987	0x4942f193	BAD_OPERATION_EXTRACT_OBJECT_2
1229123988	0x4942f194	BAD_OPERATION_EXTRACT_TYPECODE
1229123989	0x4942f195	BAD_OPERATION_EXTRACT_PRINCIPAL
1229123990	0x4942f196	BAD_OPERATION_EXTRACT_VALUE
1229123991	0x4942f197	BAD_OPERATION_GET_PRIMITIVE_TC_1
1229123992	0x4942f198	BAD_OPERATION_GET_PRIMITIVE_TC_2
1229123993	0x4942f199	BAD_OPERATION_INVOKE_NULL_PARAM_1
1229123994	0x4942f19a	BAD_OPERATION_INVOKE_NULL_PARAM_2
1229123995	0x4942f19b	BAD_OPERATION_INVOKE_DEFAULT_1
1229123996	0x4942f19c	BAD_OPERATION_INVOKE_DEFAULT_2
1229123997	0x4942f19d	BAD_OPERATION_UNKNOWN_BOOTSTRAP_METHOD
1229124097	0x4942f201	NULL_PARAM_1
1229124098	0x4942f202	NULL_PARAM_2
1229124099	0x4942f203	NULL_PARAM_3
1229124100	0x4942f204	NULL_PARAM_4
1229124101	0x4942f205	NULL_PARAM_5
1229124102	0x4942f206	NULL_PARAM_6
1229124103	0x4942f207	NULL_PARAM_7
1229124104	0x4942f208	NULL_PARAM_8
1229124105	0x4942f209	NULL_PARAM_9
1229124106	0x4942f20a	NULL_PARAM_10
1229124107	0x4942f20b	NULL_PARAM_11
1229124108	0x4942f20c	NULL_PARAM_12
1229124109	0x4942f20d	NULL_PARAM_13
1229124110	0x4942f20e	NULL_PARAM_14
1229124111	0x4942f20f	NULL_PARAM_15
1229124112	0x4942f210	NULL_PARAM_16
1229124113	0x4942f211	NULL_PARAM_17
1229124114	0x4942f212	NULL_PARAM_18
1229124115	0x4942f213	NULL_PARAM_19
1229124116	0x4942f214	NULL_PARAM_20
1229124117	0x4942f215	NULL_IOR_OBJECT
1229124118	0x4942f216	NULL_SC_DATA
1229124126	0x4942f21e	BAD_SERVANT_TYPE
1229124127	0x4942f21f	BAD_EXCEPTION
1229124128	0x4942f220	BAD_MODIFIER_LIST
1229124129	0x4942f221	NULL_PROP_MGR
1229124130	0x4942f222	INVALID_PROPERTY
1229124131	0x4942f223	ORBINITREF_FORMAT
1229124132	0x4942f224	ORBINITREF_MISSING_OBJECTURL
1229124133	0x4942f225	ORBDEFAULTINITREF_FORMAT
1229124134	0x4942f226	ORBDEFAULTINITREF_VALUE
1229124135	0x4942f227	OBJECTKEY_SERVERUUID_LENGTH
1229124136	0x4942f228	OBJECTKEY_SERVERUUID_NULL
1229124139	0x4942f22b	NULL_OBJECT_IOR
1229124225	0x4942f281	TYPECODEIMPL_CTOR_MISUSE_1
1229124226	0x4942f282	TYPECODEIMPL_CTOR_MISUSE_2
1229124227	0x4942f283	TYPECODEIMPL_NULL_INDIRECTTYPE
1229124228	0x4942f284	TYPECODEIMPL_RECURSIVE_TYPECODES
1229124235	0x4942f28b	TYPECODEIMPL_KIND_INVALID_1
1229124236	0x4942f28c	TYPECODEIMPL_KIND_INVALID_2
1229124237	0x4942f28d	TYPECODEIMPL_NATIVE_1

```
1229124238   0x4942f28e   TYPECODEIMPL_NATIVE_2
1229124239   0x4942f28f   TYPECODEIMPL_NATIVE_3
1229124240   0x4942f290   TYPECODEIMPL_KIND_INDIRECT_1
1229124241   0x4942f291   TYPECODEIMPL_KIND_INDIRECT_2
1229124242   0x4942f292   TYPECODEIMPL_NULL_TYPECODE
1229124243   0x4942f293   TYPECODEIMPL_BODY_OF_TYPECODE
1229124244   0x4942f294   TYPECODEIMPL_KIND_RECURSIVE_2
1229124245   0x4942f295   TYPECODEIMPL_COMPLEX_DEFAULT_1
1229124246   0x4942f296   TYPECODEIMPL_COMPLEX_DEFAULT_3
1229124247   0x4942f297   TYPECODEIMPL_INDIRECTION
1229124248   0x4942f298   TYPECODEIMPL_SIMPLE_DEFAULT
1229124249   0x4942f299   TYPECODEIMPL_NOT_CDROS
1229124353   0x4942f301   CONNECT_FAILURE_1
1229124354   0x4942f302   CONNECT_FAILURE_2
1229124355   0x4942f303   CONNECT_FAILURE_3
1229124356   0x4942f304   CONNECT_FAILURE_4
1229124357   0x4942f305   CONN_PURGE_REBIND
1229124358   0x4942f306   CONN_PURGE_ABORT
1229124359   0x4942f307   CONN_NOT_ESTABLISH
1229124360   0x4942f308   CONN_CLOSE_REBIND
1229124368   0x4942f310   WRITE_ERROR_SEND
1229124376   0x4942f318   GET_PROPERTIES_ERROR
1229124384   0x4942f320   BOOTSTRAP_SERVER_NOT_AVAIL
1229124392   0x4942f328   INVOKE_ERROR
1229124481   0x4942f381   BAD_HEX_DIGIT
1229124482   0x4942f382   BAD_STRINGIFIED_IOR_LEN
1229124483   0x4942f383   BAD_STRINGIFIED_IOR
1229124485   0x4942f385   BAD_MODIFIER_1
1229124486   0x4942f386   BAD_MODIFIER_2
1229124488   0x4942f388   CODESET_INCOMPATIBLE
1229124490   0x4942f38a   LONG_DOUBLE_NOT_IMPLEMENTED_1
1229124491   0x4942f38b   LONG_DOUBLE_NOT_IMPLEMENTED_2
1229124492   0x4942f38c   LONG_DOUBLE_NOT_IMPLEMENTED_3
1229124496   0x4942f390   COMPLEX_TYPES_NOT_IMPLEMENTED
1229124497   0x4942f391   VALUE_BOX_NOT_IMPLEMENTED
1229124865   0x4942f501   TRANS_NS_CANNOT_CREATE_INITIAL_NC_SYS
1229124866   0x4942f502   TRANS_NS_CANNOT_CREATE_INITIAL_NC
1229124867   0x4942f503   GLOBAL_ORB_EXISTS
1229124868   0x4942f504   PLUGINS_ERROR
1229124993   0x4942f581   BAD_REPLYSTATUS
1229124994   0x4942f582   PEEKSTRING_FAILED
1229124995   0x4942f583   GET_LOCAL_HOST_FAILED
1229124996   0x4942f584   CREATE_LISTENER_FAILED
1229124997   0x4942f585   BAD_LOCATE_REQUEST_STATUS
1229124998   0x4942f586   STRINGIFY_WRITE_ERROR
1229125000   0x4942f588   BAD_GIOP_REQUEST_TYPE_1
1229125001   0x4942f589   BAD_GIOP_REQUEST_TYPE_2
1229125002   0x4942f58a   BAD_GIOP_REQUEST_TYPE_3
1229125003   0x4942f58b   BAD_GIOP_REQUEST_TYPE_4
1229125005   0x4942f58d   NULL_ORB_REFERENCE
1229125006   0x4942f58e   NULL_NAME_REFERENCE
1229125008   0x4942f590   ERROR_UNMARSHALING_USEREXC
1229125009   0x4942f591   SUBCONTRACTREGISTRY_ERROR
1229125010   0x4942f592   LOCATIONFORWARD_ERROR
1229125011   0x4942f593   BAD_READER_THREAD
```

1229125013	0x4942f595	BAD_REQUEST_ID
1229125014	0x4942f596	BAD_SYSTEMEXCEPTION
1229125015	0x4942f597	BAD_COMPLETION_STATUS
1229125016	0x4942f598	INITIAL_REF_ERROR
1229125017	0x4942f599	NO_CODEC_FACTORY
1229125018	0x4942f59a	BAD_SUBCONTRACT_ID
1229125019	0x4942f59b	BAD_SYSTEMEXCEPTION_2
1229125020	0x4942f59c	NOT_PRIMITIVE_TYPECODE
1229125021	0x4942f59d	BAD_SUBCONTRACT_ID_2
1229125022	0x4942f59e	NAMING_CTX_REBIND_ALREADY_BOUND
1229125023	0x4942f59f	NAMING_CTX_REBINDCTX_ALREADY_BOUND
1229125024	0x4942f5a0	NAMING_CTX_BAD_BINDINGTYPE
1229125025	0x4942f5a1	NAMING_CTX_RESOLVE_CANNOT_NARROW_TO_CTX
1229125032	0x4942f5a8	TRANS_NC_BIND_ALREADY_BOUND
1229125033	0x4942f5a9	TRANS_NC_LIST_GOT_EXC
1229125034	0x4942f5aa	TRANS_NC_NEWCTX_GOT_EXC
1229125035	0x4942f5ab	TRANS_NC_DESTROY_GOT_EXC
1229125042	0x4942f5b2	INVALID_CHAR_CODESET_1
1229125043	0x4942f5b3	INVALID_CHAR_CODESET_2
1229125044	0x4942f5b4	INVALID_WCHAR_CODESET_1
1229125045	0x4942f5b5	INVALID_WCHAR_CODESET_2
1229125046	0x4942f5b6	GET_HOST_ADDR_FAILED
1229125047	0x4942f5b7 x	REACHED_UNREACHABLE_PATH
1229125048	0x4942f5b8	PROFILE_CLONE_FAILED
1229125049	0x4942f5b9	INVALID_LOCATE_REQUEST_STATUS
1229125512	0x4942f788	BAD_CODE_SET
1229125520	0x4942f790	INV_RMI_STUB
1229125521	0x4942f791	INV_LOAD_STUB
1229125522	0x4942f792	INV_OBJ_IMPLEMENTATION
1229125523	0x4942f793	OBJECTKEY_NOMAGIC
1229125524	0x4942f794	OBJECTKEY_NOSCID
1229125525	0x4942f795	OBJECTKEY_NOSERVERID
1229125526	0x4942f796	OBJECTKEY_NOSERVERUUID
1229125527	0x4942f797	OBJECTKEY_SERVERUUIDKEY
1229125762	0x4942f882	UNSPECIFIED_MARSHAL_1
1229125763	0x4942f883	UNSPECIFIED_MARSHAL_2
1229125764	0x4942f884	UNSPECIFIED_MARSHAL_3
1229125765	0x4942f885	UNSPECIFIED_MARSHAL_4
1229125766	0x4942f886	UNSPECIFIED_MARSHAL_5
1229125767	0x4942f887	UNSPECIFIED_MARSHAL_6
1229125768	0x4942f888	UNSPECIFIED_MARSHAL_7
1229125769	0x4942f889	UNSPECIFIED_MARSHAL_8
1229125770	0x4942f88a	UNSPECIFIED_MARSHAL_9
1229125771	0x4942f88b	UNSPECIFIED_MARSHAL_10
1229125772	0x4942f88c	UNSPECIFIED_MARSHAL_11
1229125773	0x4942f88d	UNSPECIFIED_MARSHAL_12
1229125774	0x4942f88e	UNSPECIFIED_MARSHAL_13
1229125775	0x4942f88f	UNSPECIFIED_MARSHAL_14
1229125776	0x4942f890	UNSPECIFIED_MARSHAL_15
1229125777	0x4942f891	UNSPECIFIED_MARSHAL_16
1229125778	0x4942f892	UNSPECIFIED_MARSHAL_17
1229125779	0x4942f893	UNSPECIFIED_MARSHAL_18
1229125780	0x4942f894	UNSPECIFIED_MARSHAL_19
1229125781	0x4942f895	UNSPECIFIED_MARSHAL_20
1229125782	0x4942f896	UNSPECIFIED_MARSHAL_21

1229125783	0x4942f897	UNSPECIFIED_MARSHAL_22
1229125784	0x4942f898	UNSPECIFIED_MARSHAL_23
1229125785	0x4942f899	UNSPECIFIED_MARSHAL_24
1229125786	0x4942f89a	UNSPECIFIED_MARSHAL_25
1229125787	0x4942f89b	UNSPECIFIED_MARSHAL_26
1229125788	0x4942f89c	UNSPECIFIED_MARSHAL_27
1229125789	0x4942f89d	UNSPECIFIED_MARSHAL_28
1229125790	0x4942f89e	UNSPECIFIED_MARSHAL_29
1229125791	0x4942f89f	UNSPECIFIED_MARSHAL_30
1229125792	0x4942f8a0	UNSPECIFIED_MARSHAL_31
1229125793	0x4942f8a1	UNSPECIFIED_MARSHAL_32
1229125794	0x4942f8a2	UNSPECIFIED_MARSHAL_33
1229125795	0x4942f8a3	UNSPECIFIED_MARSHAL_34
1229125796	0x4942f8a4	UNSPECIFIED_MARSHAL_35
1229125797	0x4942f8a5	UNSPECIFIED_MARSHAL_36
1229125798	0x4942f8a6	UNSPECIFIED_MARSHAL_37
1229125799	0x4942f8a7	UNSPECIFIED_MARSHAL_38
1229125800	0x4942f8a8	UNSPECIFIED_MARSHAL_39
1229125801	0x4942f8a9	UNSPECIFIED_MARSHAL_40
1229125802	0x4942f8aa	UNSPECIFIED_MARSHAL_41
1229125803	0x4942f8ab	UNSPECIFIED_MARSHAL_42
1229125804	0x4942f8ac	UNSPECIFIED_MARSHAL_43
1229125805	0x4942f8ad	UNSPECIFIED_MARSHAL_44
1229125806	0x4942f8ae	UNSPECIFIED_MARSHAL_45
1229125807	0x4942f8af	UNSPECIFIED_MARSHAL_46
1229125808	0x4942f8b0	UNSPECIFIED_MARSHAL_47
1229125809	0x4942f8b1	UNSPECIFIED_MARSHAL_48
1229125810	0x4942f8b2	UNSPECIFIED_MARSHAL_49
1229125811	0x4942f8b3	UNSPECIFIED_MARSHAL_50
1229125812	0x4942f8b4	UNSPECIFIED_MARSHAL_51
1229125813	0x4942f8b5	UNSPECIFIED_MARSHAL_52
1229125818	0x4942f8ba	UNSPECIFIED_MARSHAL_57
1229125819	0x4942f8bb	UNSPECIFIED_MARSHAL_58
1229125820	0x4942f8bc	UNSPECIFIED_MARSHAL_59
1229125821	0x4942f8bd	UNSPECIFIED_MARSHAL_60
1229125822	0x4942f8be	UNSPECIFIED_MARSHAL_61
1229125823	0x4942f8bf	UNSPECIFIED_MARSHAL_62
1229125824	0x4942f8c0	UNSPECIFIED_MARSHAL_63
1229125825	0x4942f8c1	READ_OBJECT_EXCEPTION_1
1229125826	0x4942f8c2	READ_OBJECT_EXCEPTION_2
1229125828	0x4942f8c4	UNSUPPORTED_IDLTYPE
1229125842	0x4942f8d2	DSI_RESULT_EXCEPTION
1229125844	0x4942f8d4	IIOPINPUTSTREAM_GROW
1229125847	0x4942f8d7	NO_CHAR_CONVERTER_1
1229125848	0x4942f8d8	NO_CHAR_CONVERTER_2
1229125849	0x4942f8d9	CHARACTER_MALFORMED_1
1229125850	0x4942f8da	CHARACTER_MALFORMED_2
1229125851	0x4942f8db	CHARACTER_MALFORMED_3
1229125852	0x4942f8dc	CHARACTER_MALFORMED_4
1229125854	0x4942f8de	INCORRECT_CHUNK_LENGTH
1229125856	0x4942f8e0	CHUNK_OVERFLOW
1229125858	0x4942f8e2	CANNOT_GROW
1229125859	0x4942f8e3	CODESET_ALREADY_SET
1229125860	0x4942f8e4	REQUEST_CANCELLED
1229125861	0x4942f8e5	WRITE_TO_STREAM_1

1229125862	0x4942f8e6	WRITE_TO_STREAM_2
1229125863	0x4942f8e7	WRITE_TO_STREAM_3
1229125864	0x4942f8e8	WRITE_TO_STREAM_4
1229125889	0x4942f901	DSI_NOT_IMPLEMENTED
1229125890	0x4942f902	GETINTERFACE_NOT_IMPLEMENTED
1229125891	0x4942f903	SEND_DEFERRED_NOTIMPLEMENTED
1229125893	0x4942f905	ARGUMENTS_NOTIMPLEMENTED
1229125894	0x4942f906	RESULT_NOTIMPLEMENTED
1229125895	0x4942f907	EXCEPTIONS_NOTIMPLEMENTED
1229125896	0x4942f908	CONTEXTLIST_NOTIMPLEMENTED
1229125902	0x4942f90e	CREATE_OBJ_REF_BYTE_NOTIMPLEMENTED
1229125903	0x4942f90f	CREATE_OBJ_REF_IOR_NOTIMPLEMENTED
1229125904	0x4942f910	GET_KEY_NOTIMPLEMENTED
1229125905	0x4942f911	GET_IMPL_ID_NOTIMPLEMENTED
1229125906	0x4942f912	GET_SERVANT_NOTIMPLEMENTED
1229125907	0x4942f913	SET_ORB_NOTIMPLEMENTED
1229125908	0x4942f914	SET_ID_NOTIMPLEMENTED
1229125909	0x4942f915	GET_CLIENT_SUBCONTRACT_NOTIMPLEMENTED
1229125913	0x4942f919	CONTEXTIMPL_NOTIMPLEMENTED
1229125914	0x4942f91a	CONTEXT_NAME_NOTIMPLEMENTED
1229125915	0x4942f91b	PARENT_NOTIMPLEMENTED
1229125916	0x4942f91c	CREATE_CHILD_NOTIMPLEMENTED
1229125917	0x4942f91d	SET_ONE_VALUE_NOTIMPLEMENTED
1229125918	0x4942f91e	SET_VALUES_NOTIMPLEMENTED
1229125919	0x4942f91f	DELETE_VALUES_NOTIMPLEMENTED
1229125920	0x4942f920	GET_VALUES_NOTIMPLEMENTED
1229125922	0x4942f922	GET_CURRENT_NOTIMPLEMENTED_1
1229125923	0x4942f923	GET_CURRENT_NOTIMPLEMENTED_2
1229125924	0x4942f924	CREATE_OPERATION_LIST_NOTIMPLEMENTED_1
1229125925	0x4942f925	CREATE_OPERATION_LIST_NOTIMPLEMENTED_2
1229125926	0x4942f926	GET_DEFAULT_CONTEXT_NOTIMPLEMENTED_1
1229125927	0x4942f927	GET_DEFAULT_CONTEXT_NOTIMPLEMENTED_2
1229125928	0x4942f928	SHUTDOWN_NOTIMPLEMENTED
1229125929	0x4942f929	WORK_PENDING_NOTIMPLEMENTED
1229125930	0x4942f92a	PERFORM_WORK_NOTIMPLEMENTED
1229125931	0x4942f92b	COPY_TK_ABSTRACT_NOTIMPLEMENTED
1229125932	0x4942f92c	PI_CLIENT_GET_POLICY_NOTIMPLEMENTED
1229125933	0x4942f92d	PI_SERVER_GET_POLICY_NOTIMPLEMENTED
1229125934	0x4942f92e	ADDRESSING_MODE_NOTIMPLEMENTED_1
1229125935	0x4942f92f	ADDRESSING_MODE_NOTIMPLEMENTED_2
1229125936	0x4942f930	SET_OBJECT_RESOLVER_NOTIMPLEMENTED
1229126017	0x4942f981	MARSHAL_NO_MEMORY_1
1229126018	0x4942f982	MARSHAL_NO_MEMORY_2
1229126019	0x4942f983	MARSHAL_NO_MEMORY_3
1229126020	0x4942f984	MARSHAL_NO_MEMORY_4
1229126021	0x4942f985	MARSHAL_NO_MEMORY_5
1229126022	0x4942f986	MARSHAL_NO_MEMORY_6
1229126023	0x4942f987	MARSHAL_NO_MEMORY_7
1229126024	0x4942f988	MARSHAL_NO_MEMORY_8
1229126025	0x4942f989	MARSHAL_NO_MEMORY_9
1229126026	0x4942f98a	MARSHAL_NO_MEMORY_10
1229126027	0x4942f98b	MARSHAL_NO_MEMORY_11
1229126028	0x4942f98c	MARSHAL_NO_MEMORY_12
1229126029	0x4942f98d	MARSHAL_NO_MEMORY_13
1229126030	0x4942f98e	MARSHAL_NO_MEMORY_14

1229126031	0x4942f98f	MARSHAL_NO_MEMORY_15
1229126032	0x4942f990	MARSHAL_NO_MEMORY_16
1229126033	0x4942f991	MARSHAL_NO_MEMORY_17
1229126034	0x4942f992	MARSHAL_NO_MEMORY_18
1229126035	0x4942f993	MARSHAL_NO_MEMORY_19
1229126036	0x4942f994	MARSHAL_NO_MEMORY_20
1229126037	0x4942f995	MARSHAL_NO_MEMORY_21
1229126038	0x4942f996	MARSHAL_NO_MEMORY_22
1229126039	0x4942f997	MARSHAL_NO_MEMORY_23
1229126040	0x4942f998	MARSHAL_NO_MEMORY_24
1229126041	0x4942f999	MARSHAL_NO_MEMORY_25
1229126042	0x4942f99a	MARSHAL_NO_MEMORY_26
1229126043	0x4942f99b	MARSHAL_NO_MEMORY_27
1229126044	0x4942f99c	MARSHAL_NO_MEMORY_28
1229126045	0x4942f99d	MARSHAL_NO_MEMORY_29
1229126046	0x4942f99e	MARSHAL_NO_MEMORY_30
1229126047	0x4942f99f	MARSHAL_NO_MEMORY_31
1229126401	0x4942fb01	RESPONSE_TIMED_OUT
1229126402	0x4942fb02	FRAGMENT_TIMED_OUT
1229126529	0x4942fb81	NO_SERVER_SC_IN_DISPATCH
1229126530	0x4942fb82	NO_SERVER_SC_IN_LOOKUP
1229126531	0x4942fb83	NO_SERVER_SC_IN_CREATE_DEFAULT_SERVER
1229126532	0x4942fb84	NO_SERVER_SC_IN_SETUP
1229126533	0x4942fb85	NO_SERVER_SC_IN_LOCATE
1229126534	0x4942fb86	NO_SERVER_SC_IN_DISCONNECT
1229126539	0x4942fb8b	ORB_CONNECT_ERROR_1
1229126540	0x4942fb8c	ORB_CONNECT_ERROR_2
1229126541	0x4942fb8d	ORB_CONNECT_ERROR_3
1229126542	0x4942fb8e	ORB_CONNECT_ERROR_4
1229126543	0x4942fb8f	ORB_CONNECT_ERROR_5
1229126544	0x4942fb90	ORB_CONNECT_ERROR_6
1229126545	0x4942fb91	ORB_CONNECT_ERROR_7
1229126546	0x4942fb92	ORB_CONNECT_ERROR_8
1229126547	0x4942fb93	ORB_CONNECT_ERROR_9
1229126548	0x4942fb94	ORB_REGISTER_1
1229126549	0x4942fb95	ORB_REGISTER_2
1229126553	0x4942fb99	ORB_REGISTER_LOCAL_1
1229126554	0x4942fb9a	ORB_REGISTER_LOCAL_2
1229126657	0x4942fc01	LOCATE_UNKNOWN_OBJECT
1229126658	0x4942fc02	BAD_SERVER_ID_1
1229126659	0x4942fc03	BAD_SERVER_ID_2
1229126660	0x4942fc04	BAD_IMPLID
1229126665	0x4942fc09	BAD_SKELETON_1
1229126666	0x4942fc0a	BAD_SKELETON_2
1229126673	0x4942fc11	SERVANT_NOT_FOUND_1
1229126674	0x4942fc12	SERVANT_NOT_FOUND_2
1229126675	0x4942fc13	SERVANT_NOT_FOUND_3
1229126676	0x4942fc14	SERVANT_NOT_FOUND_4
1229126677	0x4942fc15	SERVANT_NOT_FOUND_5
1229126678	0x4942fc16	SERVANT_NOT_FOUND_6
1229126679	0x4942fc17	SERVANT_NOT_FOUND_7
1229126687	0x4942fc1f	SERVANT_DISCONNECTED_1
1229126688	0x4942fc20	SERVANT_DISCONNECTED_2
1229127297	0x4942fe81	UNKNOWN_CORBA_EXC
1229127298	0x4942fe82	RUNTIMEEXCEPTION

```
1229127299    0x4942fe83    UNKNOWN_SERVER_ERROR
1229127300    0x4942fe84    UNKNOWN_DSI_SYSEX
1229127301    0x4942fe85    UNNOWN_IN_READ_VALUE
1229127302    0x4942fe86    UNKNOWN_CREATE_EXCEPTION_RESPONSE
1229127312    0x4942fe90    UNKNOWN_PI_EXC
1229127313    0x4942fe91    UNKNOWN_PI_EXC_2
1229127314    0x4942fe92    PI_ARGS_FAILURE
1229127315    0x4942fe93    PI_EXCEPTS_FAILURE
1229127316    0x4942fe94    PI_CONTEXTS_FAILURE
1229127317    0x4942fe95    PI_OP_CONTEXT_FAILURE
1229127326    0x4942fe9e    USER_DEFINED_ERROR
1229127327    0x4942fe9f    UNKNOWN_RUNTIME_IN_BOOTSTRAP
1229127328    0x4942fea0    UNKNOWN_THROWABLE_IN_BOOTSTRAP
1229127329    0x4942fea1    UNKNOWN_RUNTIME_IN_INSAGENT
1229127330    0x4942fea2    UNKNOWN_THROWABLE_IN_INSAGENT
1330446337    0x4f4d0001    CHARACTER_NOT_MAPPED
1330446338    0x4f4d0002    INCOMPATIBLE_VALUE_IMPL
1330446339    0x4f4d0003    NAME_ALREADY_USED_IN_THE_CONTEXT_IN_IFR
1330446340    0x4f4d0004    CANNOT_MARSHAL_LOCAL_OBJECT
1330446341    0x4f4d0005    NAME_CLASH_IN_INHERITED_CONTEXT
1330446342    0x4f4d0006    INCORRECT_TYPE_FOR_ABSTRACT_INTERFACE
1330446343    0x4f4d0007    INS_BAD_SCHEME_NAME
1330446344    0x4f4d0008    INS_BAD_ADDRESS
1330446345    0x4f4d0009    INS_BAD_SCHEME_SPECIFIC_PART
1330446346    0x4f4d000a    INS_OTHER
1330446350    0x4f4d000e    INVALID_PI_CALL
1330446351    0x4f4d000f    SERVICE_CONTEXT_ID_EXISTS
1330446359    0x4f4d0017    NO_TRANSMISSION_CODE
1330446362    0x4f4d001a    INVALID_SERVICE_CONTEXT
1330446363    0x4f4d001b    NULL_OBJECT_ON_REGISTER
1330446364    0x4f4d001c    INVALID_COMPONENT_ID
1330446365    0x4f4d001d    INVALID_IOR_PROFILE
```

If none of these steps fixes your problem, check to see if the problem has been identified and documented by looking at the available online support (hints and tips, technotes, and fixes). If you do not find your problem listed there contact IBM Support.

Message reference

You can log WebSphere Application Server system messages from a variety of sources, including application server components and applications. Messages logged by application server components and associated IBM products start with a unique message identifier that indicates the component or application that issued the message. The message identifier can be either 8 or 9 characters in length and has the form:

```
CCCC1234X
```

where:

CCCC is a four character alphabetic component or application identifier.

1234 is a fourcharacter numeric identifier used to identify the specific message for that component.

X   is an optional alphabetic severity indicator. (I=Informational, W=Warning, E=Error)

To view the messages generated by WebSphere Application Server components, select the **Quick Reference** view of this InfoCenter and expand the topic **Messages**.

CORBA minor codes

Overview

Common Object Request Broker Architecture (CORBA) is an industry-wide standard for object-oriented communication between processes, which is supported in several programming languages. Several subcomponents of WebSphere Application Server use CORBA to communicate across processes.

When a CORBA process fails, that is a request from one process to another cannot be sent, completed, or returned, a high-level exception is thrown, such as TransactionRolledBackException: CORBA TRANSACTION_ROLLEDBACK. In order to show the underlying cause of the failure, applications which use CORBA services generate minor codes, which appear in the exception stack. Look for "minor code" in the exception stack to locate these exceptions.

Minor codes used by WebSphere Application Server components

Range	Related subcomponent	Where to find details
0x49424300-0x494243FF	Security	"Security components troubleshooting tips"
0x49421050-0x4942105F, 0x49421070-0x4942107F	ORB services	"Object Request Broker component troubleshooting tips"
0x4f4d and above	Standard CORBA exceptions	http://www.omg.org
0x49421080-0x4942108F	Naming services	Javadoc for class ws.code.naming.src.com.ibm.websphere.naming.WsnCorbaMinorCodes
0x49421080-0x4942108F	Workload Management	Javadoc for class com.ibm.websphere.wlm.WsCorbaMinorCodes

Working with message logs

WebSphere Application Server can write system messages to several general purpose logs. These include the JVM logs, the process logs and the IBM service log.

The JVM logs are created by redirecting the `System.out` and `System.err` streams of the JVM to independent log files. WebSphere Application Server writes formatted messages to the `System.out` stream. In addition, applications and other code can write to these streams using the `print()` and `println()` methods defined by the streams. Some JDK built-ins such as the `printStackTrace()` method on the `Throwable` class can also write to these streams. Typically, the `System.out` log is used to monitor the health of the running application server. The `System.out` log can be used for problem determination, but it is recommended to use the IBM Service log and the advanced capabilities of the Log Analyzer instead. The `System.err` log contains exception stack trace information that is useful when performing problem analysis.

Since each application server represents a JVM, there is one set of JVM logs for each application server and all of its applications, located by default in the

`<installation_root>/logs/<server_name>` directory. In the case of a WebSphere Application Server Network Deployment configuration, JVM logs are also created for the deployment manager and each node manager, since they also represent JVMs.

The process logs are created by redirecting the `stdout` and `stderr` streams of the process to independent log files. Native code, including the Java virtual machine (JVM) itself write to these files. As a general rule, WebSphere Application Server does not write to these files. However, these logs can contain information relating to problems in native code or diagnostic information written by the JVM.

As with JVM logs, there is a set of process logs for each application server, since each JVM is an operating system process, and in the case of a WebSphere Application Server Network Deployment configuration, a set of process logs for the deployment manager and each node manager.

The IBM service log contains both the WebSphere Application Server messages that are written to the `System.out` stream and some special messages that contain extended service information that is normally not of interest, but can be important when analyzing problems. There is one service log for all WebSphere Application Server JVMs on a node, including all application servers. The IBM Service log is maintained in a binary format and requires a special tool to view. This viewer, the Log Analyzer, provides additional diagnostic capabilities. In addition, the binary format provides capabilities that are utilized by IBM support organizations.

In addition to these general purpose logs, WebSphere Application Server contains other specialized logs that are very specific in nature and are scoped to a particular component or activity. For example, the HTTP server plugin maintains a special log. Normally, these logs are not of interest, but you might be instructed to examine one or more of these logs while performing specific problem determination procedures. For details on how and when to view the plugin log, see HTTP server and Application Server are working separately, but requests are not passing from HTTP server to Application Server.

Viewing the JVM logs

The JVM logs are written as plain text files. Therefore there are no special requirements to view these logs. They are located in are located in the `<installation_directory>/logs/<applicationServerName>` directory, and by default are named `SystemOut.log` and `SystemErr.log`.

There are two techniques that you can use to view the JVM logs for an application server:
- Use the adminstrative console. This supports viewing the JVM logs from a remote machine.
- Use a text editor on the machine where the logs are stored.

Steps for this task
1. View the JVM logs from the administrative console.
   a. Start the administrative console.
   b. Click **Troubleshooting > Logging and Tracing** in the console navigation tree, then click *server* > **JVM Logs**.
   c. Select the runtime tab.
   d. Click **View** corresponding to the log you want to view.

2. View the JVM logs from the machine where they are stored.

    a. Go to the machine where the logs are stored.

    b. Open the file in a text editor or drag and drop the file into an editing and viewing program.

Interpreting the JVM logs

The JVM logs contain print data written by applications. The application can write this data written directly in the form of `System.out.print()`, `System.err.print()`, or other method calls. The application can also write data indirectly by calling a JVM function, such as an `Exception.printStackTrace()`. In addition, the `System.out` JVM log contains system messages written by the WebSphere Application Server.

You can format application data to look like WebSphere Application Server system messages by using the Installed Application Output field of the [link to ttrb_cfgmsglogs.html] JVM Logs properties panel, or as plain text with no additional formatting . WebSphere Application Server system messages are always formatted.

Depending on how the JVM log is configured, formatted messages can be written to the JVM logs in either basic or advanced format.

Message formats

Formatted messages are written to the JVM logs in one of two formats:

Basic Format
> The format used in earlier versions of WebSphere Application Server.

Advanced Format
> Extends the basic format by adding information about an event, when possible.

Basic and advanced format fields

Basic and Advanced Formats use many of the same fields and formatting techniques. The various fields that may be found in these formats follow:

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

ThreadId
> An 8 character hexidecimal value generated from the hash code of the thread that issued the message.

ShortName
> The abbreviated name of the logging component that issued the message or trace event. This is typically the class name for WebSphere Application Server internal components, but can be some other identifier for user applications.

LongName
> The full name of the logging component that issued the message or trace event. This is typically the fully qualified class name for WebSphere Application Server internal components, but can be some other identifier for user applications.

EventType

A one character field that indicates the type of the message or trace event. Message types are in upper case. Possible values include:

A   An Audit message.

I   An Informational message.

W   A Warning message.

E   An Error message.

F   A Fatal message.

O   A message that was written directly to `System.out` by the user application or internal components.

R   A message that was written directly to `System.err` by the user application or internal components.

u   A special message type used by the message logging component of the WebSphere Application Server run time.

Z   A placeholder to indicate the type was not recognized.

ClassName

The class that issued the message or trace event.

MethodName

The method that issued the message or trace event.

Organization

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

Product

The product that issued the message or trace event.

Component

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

Basic format

Message events displayed in basic format use the following format. The notation <name> indicates mandatory fields that will always appear in the basic format message. The notation [name] indicates optional or conditional fields that will be included if they can be determined.

`<timestamp><threadId><shortName><eventType>[className][methodName]<message>`

Advanced format

Message events displayed in advanced format use the following format. The notation <name> is used to indicate mandatory fields that will always appear in the advanced format for message entries. The notation [name] is used to indicate optional or conditional fields that will be included if they can be determined.

`<timestamp><threadId><eventType><UOW><source=longName>[className]`
`[methodName]<Organization><Product><Component>`
`<message>`

Configuring the JVM logs

Before you begin

Use the administrative console to configure the JVM logs for an application server. Configuration changes for the JVM logs that are made to a running application server are not applied until the next restart of the application server.

Steps for this task

1. Start the administrative console
2. Click **Troubleshooting > Logging and Tracing**, then click *server* **> JVM Logs**.
3. Select the Configuration tab.
4. Scroll through the panel to display the attributes for the stream to configure.
5. Change the appropriate configuration attributes and click **Apply**.
6. Save your configuration changes.

JVM log settings

Use this page to view and modify the settings for the Java virtual machine (JVM) `System.out` and `System.err` logs.

To view this administrative console page, click **Troubleshooting > Logs and Trace > *server* > JVM Logs**.

File Name: Specifies the name of the `System.out` file. Press the **View** button on the Runtime tab to view the contents of this file.

The file name specified on the Configuration tab must have one of the following values:

filename

> The name of a file in the file system. It is recommended that you use a fully qualified file name. If the file name is not fully qualified, it is considered to be relative to the current working directory for the server. Each stream must be configured with a dedicated file. For example, you cannot redirect both `System.out` and `System.err` to the same physical file.
>
> If the directory containing the file already exists, the user ID under which the server is running requires read/write access to the directory. If the directory does not exist, it will be created with the proper permissions. The userid under which the server is running must have authority to create the directory.

console

> This is a special file name used to redirect the stream to the corresponding process stream. If this value is specified for `System.out`, the file is redirected to `stdout`. If this value is specified for `System.err`, the file is redirected to `stderr`.

none Discards all data written to the stream. Specifying **none** is equivalent to redirecting the stream to dev/null on a Unix system.

File formatting: Specifies the format to use in saving the `System.out` file.

Log file rotation: Use this set of configuration attributes to configure the log file to be self-managing.

A self-managing log file will write messages to a file until some criteria, either time or size, is reached. At the specified time or when the file reaches the specified size, logging is temporarily suspended and the log file is closed and renamed. The new name is based on the original name of the file, plus a timestamp qualifier that indicates when the renaming occurred. Once the renaming is complete, a new,

empty log file with the original name is reopened, and logging resumes. No messages are lost as a result of the rollover, although a single message may be split across the two files.

You can only configure a log to be self-managing if the corresponding stream is redirected to a file.

File size

Selecting this attribute allows the log file to manage itself based on the size of the file. If this option is selected, the file automatically performs self-maintenance by rolling over the file when it reaches the specified maximum size.

Maximum size

This attribute specifies the maximum size in megabytes to which the file is allowed to grow.

This attribute is only valid if the File Size attribute is selected. When the file reaches this size, it is rolled over as described above.

Time Selecting this attribute allows the log file to manage itself based on the age of the file. If this option is selected, the file will roll itself over after the specified time period.

Start Time

Specifies the hour of the day, from 1 to 24, from which the periodic rollover algorithm commences. The periodic rollover algorithm uses this hour to load the algorithm at application server startup. Once started, the rollover algorithm runs without adjustment until the application server is stopped.

Rollover period

Specifies the number of hours after which the log file will be rolled over to a new filename. Valid values are from 1 to 24.

Note that you can configure a log file to roll over based on both time and size. If both **File Size** and **Time** are selected, the file is rolled over based on the criteria that is met first. For example, if the Rollover period is set to rollover every 5 hours and the Maximum File Size is set to 2 megabytes, the file will be rolled over every 5 hours, unless the log file reaches 2 megabytes in size before the time interval elapses, in which case it will be rolled over when the size limit is reached.

Maximum Number of Historical Log Files: Specifies the number of historical, or rolled over, files to keep. When a file is rolled over, the current file to which the stream is writing is closed and renamed to a name consisting of the current name plus a timestamp The stream then reopens a new file reusing the original name and continues writing. The last *maximum number of historical files* are maintained: when a historical file ages beyond that limit it is deleted.

Installed Application Output: Specifies whether the System.out or System.err print statements issued from the application code are logged and formatted.

Show application print statements

Causes application messages written to this stream using the print and println stream methods to be shown. This will have no effect on system messages written to the stream by the WebSphere Application Server.

Format print statements

Causes application messages written to this stream using the print and println stream methods to be formatted like WebSphere system messages.

Process logs

WebSphere Application Server processes contain two output streams that are accessible to native code running in the process. These streams are the stdout and stderr streams. Native code, including the JVM, may write data to these process streams. In addition, the JVM provided System.out and System.err streams can be configured to write their data to these streams also.

By default, the `stdout` and `stderr` streams are redirected to log files at application server startup, which contain text written to the `stdout` and `stderr` streams by native modules (`.dlls`, `.exes`, UNIX libraries, and other modules). By default, these files are stored as `<installation_root>/logs/<applicationServerName>/native_stderr.log` and `native_stdout.log`.

This is a change from previous versions of WebSphere Application Server, which by default had one log file for both JVM standard output and native standard output, and one log file for both JVM standard error and native error output.

Viewing the service log

The service log is a special log written in a binary format. You cannot view the log directly using a text editor. You should never directly edit the service log, as doing so will corrupt the log. To move the service log from one machine to another, you must use a mechanism like FTP, which supports binary file transfer.

You can view the service log in two ways:

- It is recommended that you use the Log Analyzer tool to view the service log. This tool provides interactive viewing and analysis capability that is helpful in identifying problems.
- If you are unable to use the Log Analyzer tool, you can use the Showlog tool to convert the contents of the service log to a text format that you can then write to a file or dump to the command shell window. The steps for using the Showlog tool are described below.

Steps for this task

1. Open a shell window on the machine where the service log resides.
2. Change directory to `<install_directory>/bin` where `<install_directory>` is the fully qualified path where the WebSphere Application Server product is installed.
3. **(Optional)** Run the showlog script with no parameters to display usage instructions.

   On Windows systems, the script is named <samp>`showlog.bat` . On UNIX systems, the script is named <samp>`showlog.sh` .
4. Run the showlog script with the appropriate parameters.

   For example:

   To display the service log contents to the shell window, use the invocation `showlog service_log_filename`. If the service log is not in the default location, you must fully qualify the `service_log_filename`.

   To format and write the service log contents to a file use the invocation `showlog service_log_filename output_filename`. If the service log is not in the default location, you must fully qualify the `service_log_filename`.

Interpreting the service log

To take advantage of the Log Analyzer's browsing and analysis capabilities, start the Log Analyzer tool:

- On Unix systems: `<installation_root>/bin/waslogbr`
- On Windows systems: *installation_root*/bin/waslogbr.bat

Use the **File > open** menu item, and select the file `/logs/activity.log`. You can also browse to activity logs from other WebSphere Application Server installations, and even other versions of the product. Expand the tree of admininstrative and application server logging sessions. Uncolored records are "normal", yellow records are warnings, and pink records are errors. If you select a record, you will see its contents, including the basic error or warning message, date, time, which component logged the record, and which process (i.e., administrative server or an application server) it came from, in the upper-right hand pane.

The activity log does not analyze any other log files, such as `default_stderr.log` or `tracefile`. To analyze these records, right click on a record in the tree on the left (click on the **UnitOfWorkView** at the top to get them all), and select **analyze**. Any records with a green check mark next to them match a record in the symptom database. When you select a check-marked record, you will see and explanation of the problem in the lower-right-hand pane.

Updating the symptom database

The database of known problems and resolutions used when you click the analyze menu item is periodically enhanced as new problems come to light and new versions of the product are introduced. To ensure that you have the latest version of the database, use the **File > update database > advanced symptom database** menu item from within the log analyzer tool at least once a month. Users who have just installed the product and have never run the update should do so immediately, since extensive updates occurred since the version first released.

The knowledge base used for analysis is built gradually from problems reported by users. For a recently released version of the product, you might not find any analysis hits. However, the Log Analyzer tool still provides a way to see all error messages and warnings from all components in a convenient, user-friendly way.

Configuring the service log

Before you begin

By default, the service log is shared among all server processes for a node. The configuration values for the service log are inherited by each server process from the node configuration. You can configure a separate service log for each server process by overriding the configuration values at the server level.

Steps for this task

1. Start the administrative console.
2. Click **Troubleshooting > Logging and Tracing >** *server_name* **> IBM Service Logs**.
3. Select the **Enable** box to enable the service log, clear the check box to disable the log.
4. Set the name for the service log.

The default name is `<install_directory>/logs/activity.log`. If the name is changed, the run time requires write access to the new file, and the file must use the `.log` extension.

5. Set the maximum file size.

   Specifies the number of megabytes to which the file can grow. When the file reaches this size, it wraps, replacing the oldest data with the newest data.

6. Set the message filter level to the desired state.

7. Save the configuration.

8. Restart the server to apply the configuration changes.

IBM service log settings

Use this page to configure the IBM Service log, also known as the *activity log*.

To view this administrative console page, click **Troubleshooting > Logs and Trace > *server* > IBM Service Logs**.

Enable service log: Specifies creation of a log file by the IBM Service log.

File Name: Specifies the name of the file used by the IBM Service log.

Maximum File Size: Specifies the maximum size in megabytes of the service log file. The default value is 2 megabytes.

When this size is reached, the service log wraps in place. Note that the service log does not roll over to a new log file like the JVM logs.

Message Filtering: Specifies the message filter level to the desired state. You can use this option to filter out or suppress some message categories. This filter value is applied to all logs, not just the service log.

Enable Correlation ID: Specifies the generation of a correlation ID that is logged with each message.

You can use the correlation ID to correlate activity to a particular client request, or correlate activities on multiple application servers.

Configuration problem collection

Use this page to identify and view problems that exist in the current configuration.

To view this administrative console page, click **Troubleshooting > Configuration Problems** in the console navigation tree.

Click on the name of the configuration problem in the Configuration Problems table to see more information about the problem.

Configuration document validation

Use these fields to specify the level of validation to perform on configuration documents.

Cross validation

Enables cross validation of configuration documents.

Enabling cross validation enables comparison of configuration documents for conflicting settings.

Configuration Problems table
Displays any current configuration problems and their associated message data. For more information on the problem, click the message associated with the problem.

Message
Displays the message returned from the validator.

Explanation
A brief explanation of the problem.

Recommendation
Specifies the recommended action to correct the problem.

Target Object
Identifies the configuration object where the validation error occurred.

Severity
Indicates the severity of the configuration error, with 1 being a severe error. Severity decreases as the severity descriptor increases.

Local URI
Specifies the local URI of the configuration file where the error occurred.

Full URI
Specifies the full URI of the configuration file where the error occurred.

Validator classname
The classname of the validator reporting the problem.

Debugging with the Application Server Toolkit

The Application Server Toolkit, included with the WebSphere Application Server on a separately-installable CD, includes debugging functionality that is built on the Eclipse workbench and that includes the following:

The WebSphere Application Server debug adapter
which allows you to debug Web objects that are running on WebSphere Application Server and that you have launched in a browser. These objects include EJBs, JSPs, and servlets.

The JavaScript debug adapter
which enables server-side JavaScript debugging.

The Compiled language debugger
which allows you to detect and diagnose errors in compiled-language applications.

The Java development tools (JDT) debugger
which allows you to debug Java.

All of the debug components in the Application Server Toolkit can be used for debugging locally and for remote debugging.

To learn more about the debug components, launch the Application Server Toolkit, select **Help > Help Contents** and choose the **Debugger Guide bookshelf** entry. To learn about known limitations and problems that are associated with the Application Server Toolkit, see the Application Server Toolkit release notes.

Debugging WebSphere Application Server applications

In order to debug your application, you must first create a Java project or a project with a Java nature. You must then import the program that you want to debug into the project. By following the steps below, you can import the WebSphere Application Server examples into a Java project.

There are two debugging styles available. Step-by-step mode will prompt you whenever the server calls a method on a Web object. A dialog allows you to either step into the method or skip it. In the dialog, you can also turn off step-by-step mode. Alternatively, if you know which part of your program you want to debug, you can add breakpoints to this code and run until one of the breakpoints is encountered. Note that breakpoints work with both styles of debugging - step-by-step mode just allows you to see which Web objects are being called without having to set up breakpoints ahead of time.

You do not have to import all of your program into the project. If you do not import all of your program into the project, some of the source may not compile. You can still debug the project and most features of the debugger will work including breakpoints, stepping and viewing/modifying variables. However, the inspect and display features in the source view will not work if the source has build errors. The inspect and display features allow you to select an expression in the source view and evaluate it. You must import any source that you want to set breakpoints in.

Steps for this task

1. Create a Java Project by opening the New Project dialog.
2. select **Java** from the left side of the dialog and **Java Project** in the right side of the dialog.
3. Click **Next** and then specify a name for the project (such as WASExamples).
4. Press **Finish** to create the project.
5. Select the new project, choose **File > Import > File System**, then **Next** to open the import file system dialog.
6. Select the directory Browse pushbutton and go to the following directory: *installation_root*\installedApps*node_name* \DefaultApplication.ear\DefaultWebApplication.war.
7. Select the checkbox next to DefaultWebApplication.war in the left side of the Import dialog and then click **Finish**.

   This will import the JSPs and Java source for the examples into your project.
8. Add any JAR files needed to build to the Java Build Path.

   To do this, select **Properties** from the right-click menu. Choose the Java Build Path node and then select the Libraries tab. Use the Add External JARs pushbutton to add the following JAR files:
   - *installation_root*\installedApps*node_name*\DefaultApplication.ear\Increment.jar.

     Once you have added this JAR file, select it and use the **Attach Source** pushbutton to attach Increment.jar as the source - Increment.jar contains both the source and class files.
   - *installation_root*\lib\j2ee.jar
   - *installation_root*\lib\pagelist.jar
   - *installation_root*\\lib\webcontainer.jar

   Click **OK** when you have added all of the JARs.

9. **(Optional)** You can set some breakpoints in the source at this time if you like, however, it is not necessary as step-by-step mode will prompt you whenever the server calls a method on a Web object.

   Step-by-step mode is explained in more detail below.

10. To start debugging, you need to start the WebSphere Application Server in debug mode and make note of the JVM debug port.

    The default value of the JVM debug port is 7777.

11. Once the server is started, switch to the debug perspective by selecting **Window > Open Perspective > Debug**. You can also enable the debug launch in the Java Perspective by choosing **Window > Customize Perspective** and selecting the **Debug** and **Launch** checkboxes in the **Other** category.

12. Select the workbench toolbar **Debug** pushbutton and then select **WebSphere Application Server Debug** from the list of launch configurations. Click the **New** pushbutton to create a new configuration.

13. Give your configuration a name and select the project to debug (your new WASExamples project). Change the port number if you did not start the server on the default port (7777).

14. Click **Debug** to start debugging.

15. Load one of the examples in your browser (for example, http://localhost:9080/hitcount).

What to do next

To learn more about the debugging, launch the Application Server Toolkit, select **Help > Help Contents** and choose the **Debugger Guide bookshelf** entry. To learn about known limitations and problems that are associated with the Application Server Toolkit, see the Application Server Toolkit release notes.

Debugging Service details

Use this page to view and modify the settings used by the Debugging Service.

To view this administrative console page, click **Servers > Application Servers** > *server* > **Debugging Service**.

Startup
Specifies whether the server will attempt to start the Debug service when the server starts.

JVM debug port
Specifies the port that the Java virtual machine will listen on for debug connections.

JVM debug arguments
Specifies the debugging argument string used to start the JVM in debug mode.

Debug class filters
Specifies an array of classes to ignore during debugging. When running in step-by-step mode, the debugger whill not stop in classes that match a filter entry.

BSF debug port
Specifies the port that the BSF Debug Manager listens on.

BSF logging level
Specifies the level of logging provided by the BSF Debug Manager. The valid range is 0-3, with 3 being the highest level of logging.

Working with trace

Use trace to obtain detailed information about the execution of WebSphere Application Server components, including application servers, clients, and other processes in the environment. Trace files show the time and sequence of methods called by WebSphere Application Server base classes, and you can use these files to pinpoint the failure.

Collecting a trace is often requested by IBM technical support personnel. If you are not familiar with the internal structure of WebSphere Application Server, the trace output might not be meaningful to you.

Steps for this task

1. Configure an output destination to which trace data is sent.
2. Enable trace for the appropriate WebSphere Application Server or application components.
3. Run the application or operation to generate the trace data.
4. Analyze the trace data or forward it to the appropriate organization for analysis.

Enabling trace

Trace for an application server process is enabled while the server process runs by using the administrative console. You can configure the application server to start in a trace-enabled state by setting the appropriate configuration properties. You can only enable trace for an application client or standalone process at process startup.

Trace strings

By default, the trace service for all WebSphere Application Server components is disabled. To request a change to the current state of the trace service, a trace string is passed to the trace service. This trace string encodes the information detailing which level of trace to enable or disable and for which components.

You can type in Trace strings, or construct them with a user-friendly GUI in the administrative console. Trace strings must conform to a specific grammar for processing by the trace service. The specification of this grammar follows:

```
TRACESTRING=COMPONENT_TRACE_STRING[:COMPONENT_TRACE_STRING]*

 COMPONENT_TRACE_STRING=COMPONENT_NAME=LEVEL=STATE[,LEVEL=STATE]*

LEVEL = all | entryExit | debug | event

STATE = enabled | disabled

COMPONENT_NAME = COMPONENT | GROUP
```

The COMPONENT_NAME is the name of a component or group registered with the trace service. Typically, WebSphere Application Server components register using a fully qualified Java classname, for example com.ibm.servlet.engine.ServletEngine. In addition, you can use a wildcard character of asterisk (*) to terminate a component name and indicate multiple classes or packages. For example, use a component name of com.ibm.servlet.* to specify all components whose names begin with com.ibm.servlet.

Examples of legal trace strings include:

```
com.ibm.ejs.ras.ManagerAdmin=debug=enabled
com.ibm.ejs.ras.ManagerAdmin=all=enabled,event=disabled
com.ibm.ejs.ras.*=all=enabled
com.ibm.ejs.ras.*=all=enabled:com.ibm.ws.ras=debug=enabled,entryexit=enabled
```

Trace strings cannot contain blanks.

Trace strings are processed from left to right. Specifying a trace string like

```
abc.*=all=enabled,event=disabled
```

first enables all trace for all components whose names start with abc, then disables event tracing for those same components. This means that the trace string

```
abc.*=all=enabled,event=disabled
```

is equivalent to

```
abc.*=debug=enabled,entryexit=enabled
```

Enabling trace at server startup

The diagnostic trace configuration settings for a server process determines the initial trace state for a server process. The configuration settings are read at server startup and used to configure the trace service. You can also change many of the trace service properties or settings while the server process is running.

Steps for this task

1. Start the administrative console.
2. Click **Troubleshooting > Logging and Tracing** in the console navigation tree, then click **Server > Diagnostic Trace**.
3. Click **Configuration**.
4. Select the **Enable Trace** check box to enable trace, clear the check box to disable trace.
5. Set the trace specification to the desired state by entering the proper TraceString.
6. Select whether to direct trace ouput to either a file or an in-memory circular buffer.
7. **(Optional)** If the in-memory circular buffer is selected for the trace output set the size of the buffer, specified in thousands of entries.

   This is the maximum number of entries that will be retained in the buffer at any given time.
8. **(Optional)** If a file is selected for trace output, set the maximum size in megabytes to which the file should be allowed to grow.

   When the file reaches this size, the existing file will be closed, renamed, and a new file with the original name reopened. The new name of the file will be based upon the original name with a timestamp qualifier added to the name. In addition, specify the number of history files to keep.
9. Select the desired format for the generated trace.
10. Save the changed configuration.
11. Start the server.

Enabling trace on a running server

You can modify the trace service state that determines which components are being actively traced for a running server by using the following procedure.

Steps for this task

1. Start the administrative console.
2. Click **Troubleshooting > Logging and Tracing** in the console navigation tree, then click **server > Diagnostic Trace**.
3. Select the **Runtime** tab.
4. **(Optional)** Select the **Save Trace** check box if you want to write your changes back to the server configuration.
5. Change the existing trace state by changing the trace specification to the desired state.
6. **(Optional)** Configure the trace output if a change from the existing one is desired.
7. Click **Apply**.

Managing the application server trace service

You can manage the trace service for a server process while the server is stopped and while it is running. You can specify which components to trace, where to send trace output, the characteristics of the trace output device, and which format to generate trace output in.

Steps for this task
1. Start the administrative console.
2. Click **Troubleshooting > Logging and Tracing** in the console navigation tree, then click *server* **> Diagnostic Trace**
3. If the server is running, select the **Runtime** tab. If the server is stopped, select the **Configuration** tab.
4. **(Optional)** For a running server, check the Save trace check box to write your changes back to the server configuration.

   If Save trace is not selected, the changes you make will apply only for the life of the server process that is currently running.
5. Perform the desired operation:
   a. Enter the file name and click **Dump** to dump the in-memory circular buffer.
   b. To change the trace destination from a file to the in-memory circular buffer or to a different file, or to change from the in memory circular buffer to a file, select the appropriate radio buttons, then click Apply.
   c. To change the format in which trace output is generated, select the appropriate value from the drop-down list.

Interpreting trace output

On an application server, trace output can be directed either to a file or to an in-memory circular buffer. If trace output is directed to the in-memory circular buffer, it must be dumped to a file before it can be viewed.

On an application client or standalone process, trace output can be directed either to a file or to the process console window.

In all cases, trace output is generated as plain text in either basic, advanced or log analyzer format as specified by the user. The basic and advanced formats for trace output are similar to the basic and advanced formats that are available for the JVM message logs.

Basic and advanced format fields

Basic and Advanced Formats use many of the same fields and formatting techniques. The fields that can be used in these formats include:

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

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

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

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

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

>   a trace entry of type method entry.

<   a trace entry of type method exit.

e   a trace entry of type event.

d   a trace entry of type debug.

m   a trace entry of type dump.

u   a trace entry of type unconditional.

Z   a placeholder to indicate that the trace type was not recognized.

ClassName
The class that issued the message or trace event.

MethodName
The method that issued the message or trace event.

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

Product
The product that issued the message or trace event.

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

Basic format

Trace events displayed in basic format use the following format:

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

Advanced formats

Trace events displayed in advanced format use the following format:

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

Log analyzer

Specifying the log analyzer format allows you to open trace output using the Log Analyzer. This is useful if you are trying to correlate traces from two different server processes, because it allows you to use the Log Analyzer's merge capability).

Trace service settings

Use this page to review and modify the properties of the diagnostic trace service.

To view this page, click **Troubleshooting > Logs and Trace** > *server* > **Diagnostic trace**.

Enable Trace
Enables the trace service.

If this option is not selected, the following configuration properties will not be passed to the application server trace service at server startup.

Save Trace
Save changes made on the runtime tab to the trace configuration as well.

Select this box to copy runtime trace changes to the trace configuration settings as well. Saving these changes to the trace configuration will cause the changes to persist even if the application is restarted.

Trace Specification
Specifies tracing details.

Enter a trace string that specifies the components, packages, or groups to trace. The trace string must conform to the specific grammar described below. You can enter the trace string directly, or generate it using the graphical trace interface. Click **Modify** to start the graphical trace interface.

If you start the graphical trace interface from the configuration tab, a list of well-known components, packages, and groups is displayed. This list might not be exhaustive.

If you start the graphical trace interface from the runtime tab, the list of components, packages, and groups displays all such components currently registered on the running server.

The format of the trace specification is:

```
<component> = <trace_type>= enabled | disabled
```

where: *<component>* is the component for which to enable or disable tracing, and *<trace_type>* is the type of tracing to enable or disable. Separate multiple tracing specifications with colons (:).

Components correspond to Java packages and classes, or to collections of Java packages. Use * as a wildcard to indicate components that include all classes in all packages contained by the specified component. For example:

*　　Specifies all traceable code running in the application server, including WebSphere Application Server system code and customer code.

`com.ibm.ws.*`
　　specifies all classes whose package name begins with `com.ibm.ws`.

`com.ibm.ws.classloader.JarClassLoader`
　　Specifies only the JarClassLoader class.

For more information on trace string grammar, see the article *Enabling trace* in the WebSphere Application Server InfoCenter.

Trace Output
Specifies where trace output should be written.

The trace output can be written directly to an output file, or stored in memory and written to a file on demand using the Dump button found on the runtime page.

Memory Buffer
　　Specifies that the trace output should be written to an in-memory circular buffer. If you select this option you must specify the following parameters:

　　Maximum Buffer Size
　　　　Specifies the number of entries, in thousands, that can be cached in the buffer. When this number is exceeded, older entries are overwritten by new entries.

　　Dump File Name
　　　　The name of the file to which the memory buffer will be written when it is dumped. This option is only available from the runtime tab.

File　Specifies to write the trace output to a self-managing log file.

　　The self-managing log file writes messages to the file until a size criteria is reached. When the file reaches the specified size, logging is temporarily suspended and the log file is closed and renamed. The new name is based on the original name of the file, plus a timestamp qualifier that indicates when the renaming occurred. Once the renaming is complete, a new, empty log file with the original name is reopened, and logging resumes. No messages are lost as a result of the rollover, although a single message may be split across the two files.

If you select this option you must specify the following parameters:

Maximum File Size
　　Specifies the maximum size, in megabytes, to which the output file is allowed to grow.

　　This attribute is only valid if the File Size attribute is selected. When the file reaches this size, it is rolled over as described above.

Maximum Number of Historical Files
> Specifies the maximum number of rolled over files to keep.

File Name
> Specifies the name of the file to which the trace output is written.

Trace Output Format
Specifies the format of the trace output.

You can specify one of three levels for trace output:

Basic (Compatible)
> preserves only basic trace information. Select this option to minimize the amount of space taken up by the trace output.

Advanced
> preserved more specific trace information. Select this option to see detailed trace information for use in troubleshooting and problem determination.

Log Analyzer
> preserved trace information in a format that is compatible with the Log Analyzer tool. Select this option if you want to use the trace output as input to the Log Analyzer tool.

Logging and tracing settings

Use this page to view and configure logging and trace settings for the server.

To view this administrative console page, click **Troubleshooting > Logs and Trace** *server_name > service_type*.

Adding logging and tracing to your application

Designers and developers of applications that run with or under WebSphere Application Server, such as servlets, JSP files, enterprise beans, client applications, and their supporting classes, may find it useful to use the same facility for generating messages that WebSphere Application Server itself uses, JRas.

This approach has advantages over simply adding `System.out.println()` statements to your code:

- Your messages appear in the WebSphere Application Server standard message format with additional data, such as a date and time stamp, added automatically.
- You can more easily correlate problems and events in your own application to problems and events associated with WebSphere Application Server components.
- You can take advantage of the WebSphere Application Server log file management features.
- You can view your messages with the WebSphere Application Server user-friendly Log Analyzer tool.

Unless you choose to extend the JRas framework, using the JRas API set is only slightly more complicated than writing basic `System println()` statements.

Programming with the JRas framework

Use the JRas extensions to incorporate message logging and diagnostic trace into WebSphere Application Server applications. The JRas extensions are based on the standalone JRas logging toolkit.

<u>Steps for this task</u>

1. Retrieve a reference to the JRas manager.
2. Retrieve message and trace loggers by using methods on the returned manager.
3. Call the appropriate methods on the returned message and trace loggers to create message and trace entries, as appropriate.

Understanding the JRas facility

Developing, deploying and maintaining applications are complex tasks. For example, when a running application encounters an unexpected condition it might not be able to complete a requested operation. In such a case you might want the application to inform the administrator that the operation has failed and give information as to why. This enables the administrator to take the proper corrective action. Application developers or maintainers might need to gather detailed information relating to the execution path of a running application in order to determine the root cause of a failure that is due to a code bug. The facilities that are used for these purposes are typically referred to as message logging and diagnostic trace.

Message logging (messages) and diagnostic trace (trace) are conceptually quite similar, but do have important differences. It is important for application developers to understand these differences in order to use these tools properly. To start with, the following operational definitions of messages and trace are provided.

Message

> A message entry is an informational record intended to be viewed by end users, systems administrators and support personnel. The text of the message must be clear, concise and interpretable by an end user. Messages are typically localized, meaning they are displayed in the national language of the end user. Although the destination and lifetime of messages might be configurable, some level of message logging is always enabled in normal system operation. Message logging must be used judiciously due to both performance considerations and the size of the message repository.

Trace A trace entry is an information record that is intended to be used by service engineers or developers. As such a trace record may be considerably more complex, verbose and detailed than a message entry. Localization support is typically not used for trace entries. Trace entries may be fairly inscrutable, understandable only by the appropriate developer or service personnel. It is assumed that trace entries are not written during normal runtime operation, but may be enabled as needed to gather diagnostic information.

WebSphere Application Server provides a message logging and diagnostic trace API that can be used by applications. This API is based on the standalone JRas logging toolkit which was developed by IBM. The standalone JRas logging toolkit is a collection of interfaces and classes that provide message logging and diagnostic trace primitives. These primitives are not tied to any particular product or platform. The standalone JRas logging toolkit provides a limited amount of support (typically referred to as systems management support), including log file configuration support based on property files.

As designed, the standalone JRas logging toolkit does not contain the support required for integration into the WebSphere Application Server runtime or for usage in a J2EE environment. To overcome these limitations, WebSphere

Application Server provides a set of extension classes to address these
shortcomings. This collection of extension classes is referred to as the JRas
extensions. The JRas extensions do not modify the interfaces introduced by the
standalone JRas logging toolkit, but simply provide the appropriate
implementation classes. The conceptual structure introduced by the standalone
JRas logging toolkit is described below. It is equally applicable to the JRas
extensions.

JRas Concepts

The following is a basic overview of important concepts and constructs introduced
by the standalone JRas logging toolkit. It is not meant to be an exhaustive
overview of the capabilities of this logging toolkit, nor is it intended to be a
detailed discussion of usage or programming paradigms. More detailed
information, including code examples, is available in JRas extensions and its
subtopics, including in the Javadoc for the various interfaces and classes that make
up the logging toolkit.

Event Types
> The standalone JRas logging toolkit defines a set of event types for
> messages and a set of event types for trace. Examples of message types
> include informational, warning and error. Examples of trace types include
> entry, exit and trace.

Event Classes
> The standalone JRas logging toolkit defines both message and trace event
> classes.

Loggers
> A logger is the primary object with which the user code interacts. Two
> types of loggers are defined. These are message loggers and trace loggers.
> The set of methods on message loggers and trace loggers are different,
> since they provide different functionality. Message loggers create only
> message records and trace loggers create only trace records. Both types of
> loggers contain masks that indicates which categories of events the logger
> should process and which it should ignore. Although every JRas logger is
> defined to contain both a message and trace mask, the message logger only
> uses the message mask and the trace logger only uses the trace mask. For
> example, by setting a message logger's message mask to the appropriate
> state, it can be configured to process only Error messages and ignore
> Informational and Warning messages. Changing the state of a message
> logger's trace mask has no effect.
>
> A logger contains one or more handlers to which it forwards events for
> further processing. When the user calls a method on the logger, the logger
> will compare the event type specified by the caller to its current mask
> value. If the specified type passes the mask check, the logger will create an
> event object to capture the information relating to the event that was
> passed to the logger method. This information may include information
> such as the names of the class and method which is logging the event, a
> message and parameters to log, among others. Once the logger has created
> the event object, it forwards the event to all handlers currently registered
> with the logger.

Handlers
> A handler provides an abstraction over an output device or event
> consumer. An example is a file handler, which knows how to write an
> event to a file. The handler also contains a mask that is used to further

restrict the categories of events the handler will process. For example, a message logger may be configured to pass both warning and error events, but a handler attached to the message logger may be configured to only pass error events. Handlers also include formatters, which the handler invokes to format the data in the passed event before it is written to the output device.

Formatters

Handlers are configured with Formatters, which know how to format events of certain types. A handler may contain multiple formatters, each of which know how to format a specific class of event. The event object is passed to the appropriate formatter by the handler. The formatter returns formatted output to the handler, which then writes it to the output device.

JRas Extensions

The standalone JRas logging toolkit defines interfaces and provides a variety of concrete classes that implement these interfaces. Since the standalone JRas logging toolkit was developed as a general purpose toolkit, the implementation classes do not contain the configuration interfaces and methods necessary for use in the WebSphere Application Server product. In addition, many of the implementation classes are not written appropriately for use in a J2EE environment. To overcome these shortcomings, WebSphere Application Server provides the appropriate implementation classes that allow integration into the WebSphere Application Server environment. The collection of these implementation classes is referred to as the JRas extensions.

Usage Model

You can use the JRas extensions in three distinct operational modes:

Integrated

In this mode, message and trace records are written only to logs defined and maintained by the WebSphere Application Server runtime. This is the default mode of operation and is equivalent to the WebSphere Application Server 4.0 mode of operation.

Standalone

In this mode, message and trace records are written solely to standalone logs defined and maintained by the user. You control which categories of events are written to which logs, and the format in which entries are written. You are responsible for configuration and maintenance of the logs. Message and trace entries are not written to WebSphere Application Server runtime logs.

Combined

In this mode message and trace records are written to both WebSphere Application Server runtime logs and to standalone logs that you must define, control, and maintain. You can use filtering controls to determine which categories of messages and trace are written to which logs.

The JRas extensions are specifically targeted to an integrated mode of operation. The integrated mode of operation can be appropriate for some usage scenarios, but there many scenarios are not adequately addressed by these extensions. Many usage scenarios require a standalone or combined mode of operation instead. A set of user extension points has been defined that allow the JRas extensions to be used in either a standalone or combined mode of operations.

JRas extension classes: WebSphere Application Server provides a base set of implementation classes that collectively are referred to as the JRas extensions. Many of these classes provide the appropriate implementations of loggers, handlers and formatters for use in a WebSphere Application Server environment. As previously noted, this collection of classes is targeted at an Integrated mode of operation. If you choose to use the JRas extensions in either standalone or combined mode, you can reuse the logger and manager class provided by the extensions, but you must provide your own implementations of handlers and formatters.

WebSphere Application Server Message and Trace loggers

The message and trace loggers provided by the standalone JRas logging toolkit cannot be directly used in the WebSphere Application Server environment. The JRas extensions provide the appropriate logger implementation classes. Instances of these message and trace logger classes are obtained directly and exclusively from the WebSphere Application Server Manager class, described below. You cannot directly instantiate message and trace loggers. Obtaining loggers in any manner other than directly from the Manager is not allowed. Doing so is a direct violation of the programming model.

The message and trace loggers instances obtained from the WebSphere Application Server Manager class are subclasses of the `RASMessageLogger()` and `RASTraceLogger()` classes provided by the standalone JRas logging toolkit. The `RASMessageLogger()` and `RASTraceLogger()` classes define the set of methods that are directly available. Public methods introduced by the JRas extensions logger subclasses cannot be called directly by user code. Doing so is a violation of the programming model.

Loggers are named objects and are identified by name. When the Manager class is called to obtain a logger, the caller is required to specify a name for the logger. The Manager class maintains a name-to-logger instance mapping. Only one instance of a named logger will ever be created within the lifetime of a process. The first call to the Manager with a particular name will result in the logger being created and configured by the Manager. The Manager will cache a reference to the instance, then return it to the caller. Subsequent calls to the Manager that specify the same name will result in a reference to the cached logger being returned. Separate name spaces are maintained for message and trace loggers. This means a single name can be used to obtain both a message logger and a trace logger from the Manager, without ambiguity, and without causing a name space collision.

In general, loggers have no predefined granularity or scope. A single logger could be used to instrument an entire application. Or users may determine that having a logger per class is more desirable. Or the appropriate granularity may lie somewhere in between. Partitioning an application into logging domains is rightfully determined by the application writer.

The WebSphere Application Server logger classes obtained from the Manager are thread-safe. Although the loggers provided as part of the standalone JRas logging toolkit implement the serializable interface, in fact loggers are not serializable. Loggers are stateful objects, tied to a Java virtual machine instance and are not serializable. Attempting to serialize a logger is a violation of the programming model.

Please note that there is no provision for allowing users to provide their own logger subclasses for use in a WebSphere Application Server environment.

WebSphere Application Server handlers

WebSphere Application Server provides the appropriate handler class that is used to write message and trace events to the WebSphere Application Server runtime logs. You cannot configure the WebSphere Application Server handler to write to any other destination. The creation of a WebSphere Application Server handler is a restricted operation and not available to user code. Every logger obtained from the Manager comes preconfigured with an instance of this handler already installed. You can remove the WebSphere Application Server handler from a logger when you want to run in standalone mode. Once you have removed it, you cannot re-add the WebSphere Application Server handler to the logger from which it was removed (or any other logger). Also, you cannot directly call any method on the WebSphere Application Server handler. Attempting to create an instance of the WebSphere Application Server handler, to call methods on the WebSphere Application Server handler or to add a WebSphere Application Server handler to a logger by user code is a violation of the programming model.

WebSphere Application Server formatters

The WebSphere Application Server handler comes preconfigured with the appropriate formatter for data that is written to WebSphere Application Server logs. The creation of a WebSphere Application Server formatter is a restricted operation and not available to user code. No mechanism exists that allows the user to obtain a reference to a formatter installed in a WebSphere Application Server handler, or to change the formatter a WebSphere Application Server handler is configured to use.

WebSphere Application Server manager

WebSphere Application Server provides a Manager class located in the `com.ibm.websphere.ras` package. All message and trace loggers must be obtained from this Manager. A reference to the Manager is obtained by calling the static `Manager.getManager()` method. Message loggers are obtained by calling the `createRASMessageLogger()` method on the Manager. Trace loggers are obtained by calling the `createRASTraceLogger()` method on the Manager class.

The manager also supports a *group* abstraction that is useful when dealing with trace loggers. The group abstraction allows multiple, unrelated trace loggers to be registered as part of a named entity called a group. WebSphere Application Server provides the appropriate systems management facilities to manipulate the trace setting of a group, similar to the way the trace settings of an individual trace logger.

For example, suppose component A consist of 10 classes. Suppose each class is configured to use a separate trace logger. Suppose all 10 trace loggers in the component are registered as members of the same group (for example Component_A_Group). You can then turn on trace for a single class. Or you can turn on trace for all 10 classes in a single operation using the group name if you want a component trace. Group names are maintained within the name space for trace loggers.

Extending the JRas framework: Since the Jras extensions classes do not provide the flexibility and behavior required for many scenarios, a variety of extension points have been defined. You are allowed to write your own implementation classes to obtain the required behavior.

In general, the Jras extensions require you to call the Manager class to obtain a message logger or trace logger. No provision is made to allow you to provide your own message or trace logger subclasses. In general, user-provided extensions cannot be used to affect the integrated mode of operation.The behavior of the integrated mode of operation is solely determined by the WebSphere Application Server runtime and the JRas extensions classes.

Handlers

The standalone JRas logging toolkit defines the `RASIHandler` interface. All handlers must implement this interface. You can write your own handler classes that implement the `RASIHandler` interface. You should directly create instances of user-defined handlers and add them to the loggers obtained from the Manager.

The standalone JRas logging toolkit provides several handler implementation classes. These handler classes are inappropriate for usage in the J2EE environment. You cannot directly use or subclass any of the Handler classes provided by the standalone JRas logging toolkit. Doing so is a violation of the programming model.

Formatters

The standalone JRas logging toolkit defines the `RASIFormatter` interface. All formatters must implement this interface. You can write your own formatter classes that implement the `RASIFormatter` interface. You can only add these classes to a user-defined handler. WebSphere Application Server handlers cannot be configured to use user-defined formatters. Instead, directly create instances of your formatters and add them to the your handlers appropriately.

As with handlers, the standalone JRas logging toolkit provides several formatter implementation classes. Direct usage of these formatter classes is not supported.

Message event types

The standalone JRas toolkit defines message event types in the `RASIMessageEvent` interface. In addition, the WebSphere Application Server reserves a range of message event types for future use. The `RASIMessageEvent` interface defines three types, with values of 0x01, 0x02, and 0x04. The values 0x08 through 0x8000 are reserved for future use. You can provide your own message event types by extending this interface appropriately. User-defined message types must have a value of 0x1000 or greater.

Message loggers retrieved from the Manager have their message masks set to *pass* or process all message event types defined in the `RASIMessageEvent` interface. In order to process user-defined message types, you must manually set the message logger mask to the appropriate state by user code after the message logger has been obtained from the Manager. WebSphere Application Server does not provide any built-in systems management support for managing any message types.

Message event objects

The standalone Jras toolkit provides a `RASMessageEvent` implementation class. When a message logging method is called on the message logger, and the message type is currently enabled, the logger creates and distributes an event of this class to all handlers currently registered with that logger.

You can provide your own message event classes, but they must implement the `RASIEvent` interface. You must directly create instances of such user-defined message event classes. Once it is created, pass your message event to the message logger by calling the message logger's `fireRASEvent()` method directly. WebSphere Application Server message loggers cannot directly create instances of user-defined types in response to calling a logging method (`msg()`, `message()`...) on the logger. In addition, instances of user-defined message types are never processed by the WebSphere Application Server handler. You cannot create instances of the `RASMessageEvent` class directly.

Trace event types

The standalone JRas toolkit defines trace event types in the `RASITraceEvent` interface. You can provide your own trace event types by extending this interface appropriately. In such a case you must ensure that the values for the user-defined trace event types do not collide with the values of the types defined in the `RASITraceEvent` interface.

Trace loggers retrieved from the Manager typically have their trace masks set to reject all types. A different starting state can be specified by using WebSphere Application Server systems managment facilities. In addition, the state of the trace mask for a logger can be changed at runtime using WebSphere Application Server systems management facilities.

In order to process user-defined trace types, the trace logger mask must be manually set to the appropriate state by user code. WebSphere Application Server systems management facilities cannot be used to manage user-defined trace types, either at start time or runtime.

Trace event objects

The standalone Jras toolkit provides a `RASTraceEvent` implementation class. When a trace logging method is called on the WebSphere Application Server trace logger and the type is currently enabled, the logger creates and distributes an event of this class to all handlers currently registered with that logger.

You can provide your own trace event classes. Such trace event classes must implement the `RASIEvent` interface. You must create instances of such user-defined event classes directly. Once it is created, pass the trace event to the trace logger by calling the trace logger's `fireRASEvent()` method directly. WebSphere Application Server trace loggers cannot directly create instances of user-defined types in response to calling a trace method (`entry()`, `exit()`, `trace()`) on the trace logger. In addition, instances of user-defined trace types are never processed by the WebSphere Application Server handler. You cannot create instances of the `RASTraceEvent` class directly.

User defined types, user defined events and WebSphere Application Server

By definition, the WebSphere Application Server handler will process user-defined message or trace types, or user-defined message or trace event classes. Message and trace entries of either a user-defined type or user-defined event class cannot be written to the WebSphere Application Server runtime logs.

Writing User Extensions: General Considerations

You can configure the WebSphere Application Server to use Java 2 security to restrict access to protected resources such as the file system and sockets. Since user written extensions typically access such protected resources, user written extensions must contain the appropriate security checking calls, using AccessController doPrivileged() calls. In addition, the user written extensions must contain the appropriate policy file. In general, it is recommended that you locate user written extensions in a separate package. It is your responsibility to restrict access to the user written extensions appropriately.

Writing a handler

User written handlers must implement the `RASIHandler` interface. The `RASIHandler` interface extends the `RASIMaskChangeGenerator` interface, which extends the `RASIObject` interface. A short discussion of the methods introduced by each of these interfaces follows, along with implementation pointers. For more in depth information on any of the particular interfaces or methods, see the corresponding product Javadoc.

RASIObject interface

The `RASIObject` interface is the base interface for standalone JRas logging toolkit classes that are stateful or configurable, such as loggers, handlers and formatters.

- The standalone JRas logging tookit supports rudimentary properties-file based configuration. To implement this configuration support, the configuration state is stored as a set of key-value pairs in a properties file. The methods `public Hashtable getConfig()` and `public void setConfig(Hashtable ht)` are used to get and set the configuration state. The JRas extensions do not support properties based configuration and it is recommended that these methods be implemented as no-operations. You can implement your own properties based configuration using these methods.

- Loggers, handlers and formatters can be named objects. For example, the JRas extensions require the user to provide a name for the loggers that are retrieved from the manager. You can name your handlers. The methods `public String getName()` and `public void setName(String name)` are provided to get or set the name field. The JRas extensions currently do not call these methods on user handlers. You can implement these methods as you want, including as no operations.

- Loggers, handlers and formatters can also contain a description field. The methods `public String getDescription()` and `public void setDescription(String desc)` can be used to get or set the description field. The JRas extensions currently do not use the description field. You can implement these methods as you want, including as no operations.

- The method `public String getGroup()` is provided for usage by the `RASManager`. Since the JRas extensions provide their own Manager class, this method is never called. It is recommended you implement this as a no-operation.

RASIMaskChangeGenerator interface

The `RASIMaskChangeGenerator` interface is the interface that defines the implementation methods for filtering of events based on a mask state. This means that it is currently implemented by both loggers and handlers. By definition, an object that implements this interface contains both a message mask and a trace mask, although both need not be used. For example, message loggers contain a trace mask, but the trace mask is never used since the message logger never

generates trace events. Handlers however can actively use both mask values. For example a single handler could handle both message and trace events.

- The methods `public long getMessageMask()` and `public void setMessageMask(long mask)` are used to get or set the value of the message mask. The methods `public long getTraceMask()` and `public void setTraceMask(long mask)` are used to get or set the value of the trace mask.

In addition, this interface introduces the concept of *calling back* to interested parties when a mask changes state. The callback object must implement the `RASIMaskChangeListener` interface.

- The methods `public void addMaskChangeListener(RASIMaskChangeListener listener)` and `public void removeMaskChangeListener(RASIMaskChangeListener listener)` are used to add or remove listeners to the handler. The method `public Enumeration getMaskChangeListeners()` returns an Enumeration over the list of currently registered listeners. The method `public void fireMaskChangedEvent(RASMaskChangeEvent mc)` is used to call back all the registered listeners to inform them of a mask change event.

For efficiency reasons, the Jras extensions message and trace loggers implement the `RASIMaskChangeListener` interface. The logger implementations maintain a "composite mask" in addition to the logger's own mask. The logger's composite mask is formed by logically *or'ing* the appropriate masks of all handlers that are registered to that logger, then *and'ing* the result with the logger's own mask. For example, the message logger's composite mask is formed by or'ing the message masks of all handlers registered with that logger, then and'ing the result with the logger's own message mask.

This means that all handlers are required to properly implement these methods. In addition, when a user handler is instantiated, the logger it is to be added to should be registered with the handler using the `addMaskChangeListener()` method. When either the message mask or trace mask of the handler is changed, the logger must be called back to inform it of the mask change. This allows the logger to dynamically maintain the composite mask.

The `RASMaskChangedEvent` class is defined by the standalone JRas logging toolkit. Direct usage of that class by user code is allowed in this context.

In addition the `RASIMaskChangeGenerator` introduces the concept of caching the names of all message and trace event classes that the implementing object will process. The intent of these methods is to allow a management program such as a GUI to retrieve the list of names, introspect the classes to determine the event types that they might possibly process and display the results. The JRas extensions do not ever call these methods, so they can be implemented as no operations, if desired.

- The methods `public void addMessageEventClass(String name)` and `public void removeMessageEventClass(String name)` can be called to add or remove a message event class name from the list. The method `public Enumeration getMessageEventClasses()` will return an enumeration over the list of message event class names. Similarly, the `public void addTraceEventClass(String name)` and `public void removeTraceEventClass(String name)` can be called to add or remove a trace event class name from the list. The `method public Enumeration getTraceEventClasses()` will return an enumeration over the list of trace event class names.

RASIHandler interface

The `RASIHandler` interface introduces the methods that are specific to the behavior of a handler.

The `RASIHandler` interface as provided by the standalone JRas logging toolkit supports handlers that run in either a synchronous or asynchronous mode. In asynchronous mode, events are typically queued by the calling thread and then written by a worker thread. Since spawning of threads is not allowed in the WebSphere Application Server environment, it is expected that handlers will not queue or batch events, although this is not expressly prohibited.

- The methods `public int getMaximumQueueSize()` and `public void setMaximumQueueSize(int size)` throw `IllegalStateException` are provided to manage the maximum queue size. The method `public int getQueueSize()` is provided to query the actual queue size.
- The methods `public int getRetryInterval()` and `public void setRetryInterval(int interval)` support the notion of error retry, which again implies some type of queueing.
- The methods `public void addFormatter(RASIFormatter formatter)`, `public void removeFormatter(RASIFormatter formatter)` and `public Enumeration getFormatters()` are provided to manage the list of formatters that the handler can be configured with. Different formatters can be provided for different event classes, if appropriate.
- The methods `public void openDevice()`, `public void closeDevice()` and `public void stop()` are provided to manage the underlying device that the handler abstracts.
- The methods `public void logEvent(RASIEvent event)` and `public void writeEvent(RASIEvent event)` are provided to actually pass events to the handler for processing.

Writing a formatter

User written formatters must implement the `RASIFormatter` interface. The `RASIFormatter` interface extends the `RASIObject` interface. The implementation of the `RASIObject` interface is the same for both handlers and formatters. A short discussion of the methods introduced by the `RASIFormatter` interface follows. For more in depth information on the methods introduced by this interface, see the corresponding product javadoc.

RASIFormatter interface

- The methods `public void setDefault(boolean flag)` and `public boolean isDefault()` are used by the concrete RASHandler classes provided by the standalone JRas logging toolkit to determine if a particular formatter is the default formatter. Since these `RASHandler` classes must never be used in a WebSphere Application Server environment, the semantic significance of these methods can be determined by the user.
- The methods `public void addEventClass(String name)`, `public void removeEventClass(String name)` and `public Enumeration getEventClasses()` are provided to determine which event classes a formatter can be used to format. You can provide the appropriate implementations as you see fit.
- The method `public String format(RASIEvent event)` is called by handler objects and returns a formatted String representation of the event.

Example: user written handler: The following is a very simple sample of a Handler class that writes formatted events to a file. This class is functional, but is intended solely to demonstrate concepts. For simplicity and clarity, much code

(including appropriate boundary condition checking logic) has been ignored. This sample is not intended to be an example of good programming practice.

```
package com.ibm.ws.ras.test.user;

import com.ibm.ras.*;
import java.io.*;
import java.util.*;
import java.security.AccessController;
import java.security.PrivilegedAction;
import java.security.PrivilegedExceptionAction;
import java.security.PrivilegedActionException;

    /**
     * The <code>SimpleFileHandler</code> is a class that implements the
     * {{@link RASIHandler} interface. It is a simple
     * Handler that writes to a file. The name of the file must be specified in the constructor.
     * <p>
     * If the file includes a path, the path separator may be a front-slash ('/')
     * or the platform-specific path separator character.  For example:
     *
     * /Dir1/Dir2/Dir3/MyStuff.log
     *
     */

public class SimpleFileHandler implements RASIHandler
{
  /**
     * A public boolean that can be inspected by the caller to determine if an error has
     * occurred during an operation.
     * This boolean can only be changed when the device synchronizer is held.
     */
  public boolean errorHasOccurred = false;
  /**
     * The name of the Handler
     */
  private String ivName = "";

  /**
     * The message mask which determines the types of messages that will be processed.
     */
  private long ivMessageMask;



  /**
     * The trace mask which determines the types of trace points that will be processed.
     */
  private long ivTraceMask;

  /**
     * The names of the message event classes which this object processes.
     */
  private Vector ivMessageEventClasses;

  /**
     * The names of the trace event classes which this object processes.
     */
  private Vector ivTraceEventClasses;

  /**
     * The set of {@link RASIMaskChangeListener} which want to be informed of changes to the
     * <code>RASIMaskChangeGenerator</code> message or trace mask configuration.
     */
  private Vector ivMaskChangeListeners;

  /**
```

```
 * The fully-qualified, normalized name of the file to which the log entries are written.
 */
private String ivFqFileName;

/**
 * A boolean flag which indicates whether the device to which this handler sends log
 * entries is open.  It is set to true when the device is open and false otherwise.
 */
private boolean ivDeviceOpen = false;

/**
 * A Hashtable of RASIFormatters keyed by the name of the event class they format.
 * Each event type can have exactly
 * one formatter. Different event classes can have different formatters.
 */
private Hashtable ivFormatters;

/**
 * An object on which the {@link #closeDevice closeDevice} and
 * {@link #writeEvent writeEvent} methods can synchronize.
 */
private Object ivDeviceLock = new Object();

/**
 * The stream to which formatted log events are written. This stream will wrap a file.
 */
private PrintWriter ivWriter = null;

/**
 * Create a SimpleFileHandler.
 * <p>
 * The constructor will attempt to open a stream in append mode over the specified file.
 * If the operation does not complete
 * successfully, the errorHasOccurred boolean is set  to true. If no exceptions are thrown
 * by this constructor and the
 * errorHasOccurred booleans state is false, the stream is open and the handler is usable.
 * <p>
 * @param name the name assigned to this handler object. Null is tolerated.
 * @param fileName a non-null file name. Caller must guarantee this name is not null.
 * A  fully qualified file name is preferred.
 */
public SimpleFileHandler(String name, String fileName) throws Exception  {
  setName(name);
  ivMessageMask = RASIMessageEvent.DEFAULT_MESSAGE_MASK;
  ivTraceMask = RASITraceEvent.DEFAULT_TRACE_MASK;
  // Allocate the Hashtables and Vectors required.
  ivMaskChangeListeners = new Vector();
  ivMessageEventClasses = new Vector();
  ivTraceEventClasses = new Vector();
  ivFormatters = new Hashtable();
  // Add the default event classes that this handler will process
  addMessageEventClass("com.ibm.ras.RASMessageEvent");
  addTraceEventClass("com.ibm.ras.RASTraceEvent");

  // Get the fully qualified, normalized file name. Open the stream
  File x = new File(fileName);
  ivFqFileName = x.getAbsolutePath();
  openDevice();
}

////////////////////////////////////////////////////////////////////
//
// The following methods are required by the RASIObject interface
//
////////////////////////////////////////////////////////////////////

/**
```

```
 * Return this objects configuration as a set of Properties in a Hashtable.
 * <p>
 * This handler does not support properties-based configuration. Therefore a call to
 * this method always returns null
 * @return null is always returned.
 */
public Hashtable getConfig()  {
  return null;
}

/**
 * Set this objects configuration from the properties in the specified Hashtable.
 * <p>
 * This handler does not support properties-based configuration.
 * This method is a no-operation.
 * @param hashTable a Hashtable containing the properties. Input is ignored.
 */
public void setConfig(Hashtable ht)  {
  return;
}

/**
 * Return the name by which this object is known.
 * <p>
 * @return a String containing the name of this object, or an empty string ("") if
 * the name has not been set.
 */
public String getName()  {
  return ivName;
}

/**
 * Set the name by which this object is known.  If the specified name is
 * <code>null</code>, the current name is not changed.
 * <p>
 * @param name The new name for this object. Null is tolerated.
 */
public void setName(String name)    {
  if (name != null)
    ivName = name;
}

/**
 * Return the description field of this object.
 * <p>
 * This handler does not use a description field. An empty String is always returned.
 * <p>
 * @return an empty String.
 */
public String getDescription()  {
  return "";
}

/**
 * Set the description field for this object.
 * <p>
 * This handler does not use a description field. Input is ignored and
 * this method does nothing.
 * <p>
 * @param desc The description of this object. Input is ignored.
 */
public void setDescription(String desc)  {
  return;
}

/**
 * Return the name of the {@link com.ibm.ras.mgr.RASManager RASManager} group
```

```
 * with which this object is associated.  This method is only used by the  RAS Manager.
 * <p>
 * This object does not support RASManager configuration. Null is always returned.
 * @return null is always returned.
 */
public String getGroup()  {
  return null;
}

////////////////////////////////////////////////////////////
//
// Methods required by the RASIMaskChangeGenerator interface
//
////////////////////////////////////////////////////////////

/**
 * Return the message mask which defines the set of message types that will be
 * processed by this Handler. The set of possible
 * types is identified in the {@link RASIMessageEvent}
 * <code>TYPE_XXXX</code> constants.
 * <p>
 * @return The current message mask.
 */
public long getMessageMask()    {
  return ivMessageMask;
}

/**
 * Set the message mask which defines the set of message types that will be
 * processed by this Handler. The set of possible
 * types is identified in the {@link RASIMessageEvent}
 * <code>TYPE_XXXX</code> constants.
 * The mask value is not validated against these types.
 * <p>
 * @param mask The message mask.
 */
public void setMessageMask(long mask)    {
  RASMaskChangeEvent mc = new RASMaskChangeEvent(this, ivMessageMask, mask, true);
  ivMessageMask = mask;
  fireMaskChangedEvent(mc);
}

/**
 * Return the trace mask which defines the set of trace types that will be
 * processed by this Handler. The set of possible
 * types is identified in the {@link RASITraceEvent}
 * <code>TYPE_XXXX</code>
 * constants.
 * <p>
 * @return The current trace mask.
 */
public long getTraceMask()    {
  return ivTraceMask;
}

/**
 * Set the trace mask which defines the set of trace types that will be
 * processed by this Handler. The set of possible types
 * is identified in the {@link RASITraceEvent}
 * <code>TYPE_XXXX</code> constants.
 * The specified trace mask value is not validated.
 * <p>
 * @param mask The trace mask.
 */
public void setTraceMask(long mask)     {
  RASMaskChangeEvent mc = new RASMaskChangeEvent(this, ivTraceMask, mask, false);
  ivTraceMask = mask;
```

Chapter 4. Diagnosing and fixing problems   **267**

```
      fireMaskChangedEvent(mc);
    }

    /**
     * Add a {@link RASIMaskChangeListener} object to the set of listeners which wish to
     * be identified of a change in the message
     * or trace mask configuration.  If the specified listener is
     * <code>null</code> or is already registered, this method does nothing.
     * <p>
     * @param listener The mask change listener.
     */
    public void addMaskChangeListener(RASIMaskChangeListener listener)  {
      if (listener != null && (!ivMaskChangeListeners.contains(listener)))
        ivMaskChangeListeners.addElement(listener);
    }

    /**
     * Remove a {@link RASIMaskChangeListener} object from the list of registered listeners
     * that wish to be informed of changes
     * in the message or trace mask configuration. If the  listener is
     * <code>null</code> or is not registered, this method does nothing.
     * <p>
     * @param listener The mask change listener.
     */
    public void removeMaskChangeListener(RASIMaskChangeListener listener)  {
      if (listener != null && ivMaskChangeListeners.contains(listener))
        ivMaskChangeListeners.removeElement(listener);
    }

    /**
     * Return an enumeration over the set of listeners currently registered to be
     * informed of changes in the message or trace mask configuration.
     * <p>
     * @return An Enumeration of mask change listeners. If no listeners are registered,
     * the Enumeration is empty.
     */
    public Enumeration getMaskChangeListeners()  {
      return ivMaskChangeListeners.elements();
    }

    /**
     * Inform all registered <code>RASIMaskChangeListener</code>s that
     * the message or trace mask has been changed.
     *<p>
     * @param mc A mask change event, indicating what has changed.
     */
    public void fireMaskChangedEvent(RASMaskChangeEvent mc)  {
      RASIMaskChangeListener c;
      Enumeration e = getMaskChangeListeners();
      while (e.hasMoreElements()) {
        c = (RASIMaskChangeListener) e.nextElement();
        c.maskValueChanged(mc);
      }
    }

    /**
     * Add the name of a message event class to the list of message event classes
     * which this  handler processes. If the specified
     * event class is null or is already  registered, this method does nothing.
     * <p>
     * @param name The event class name.
     */
    public void addMessageEventClass(String name)  {
      if (name != null && (! ivMessageEventClasses.contains(name)))
        ivMessageEventClasses.addElement(name);
    }
```

```
/**
 * Remove the name of a message event class from the list of names of classes
 * which this handler processes. If the specified event
 * class is null or is not registered, this method does nothing.
 * <p>
 * @param name The event class name.
 */
public void removeMessageEventClass(String name)  {
  if ((name != null) && (ivMessageEventClasses.contains(name)))
    ivMessageEventClasses.removeElement(name);
}

/**
 * Return an enumeration over the set of names of the message event classes
 * which this handler processes.
 * <p>
 * @return An Enumeration of RAS event class names.  If no event classes
 * are registered, the Enumeration is empty.
 */
public Enumeration getMessageEventClasses()  {
  return ivMessageEventClasses.elements();
}

/**
 * Add the name of a trace event class to the list of trace event classes
 * which this handler processes.  If the specified event
 * class is null or is already  registered, this method does nothing.
 * <p>
 * @param name The event class name.
 */
public void addTraceEventClass(String name)  {
  if ((name != null) && (!ivTraceEventClasses.contains(name)))
    ivTraceEventClasses.addElement(name);
}

/**
 * Remove the name of a trace event class from the list of names of classes
 * which this handler processes. If the
 * specified event class is null or is not registered, this method does
 * nothing.
 * <p>
 * @param name The event class name.
 */
public void removeTraceEventClass(String name)  {
  if ((name != null) && (ivTraceEventClasses.contains(name)))
    ivTraceEventClasses.removeElement(name);
}

/**
 * Return an enumeration over the set of names of the trace event classes
 * which this handler processes
 * <p>
 * @return An Enumeration of RAS event class names.  If no event classes are
 * registered, the Enumeration is empty.
 */
public Enumeration getTraceEventClasses()  {
  return ivTraceEventClasses.elements();
}

///////////////////////////////////////////////////////
//
// Methods required by the RASIHandler interface
//
///////////////////////////////////////////////////////

/**
 * Return the maximum number of {@link RASIEvent RASIEvents} which this handler
```

```
   * will queue before writing.
   * <p>
   * In the WebSphere Application Server environment, handlers may not start
   * threads.
   * All writes will be done
   * synchonously and never queued. This handler does not queue events for later
   * retry if a write operation fails.
   * <p>
   * @return zero is always returned.
   */
public int getMaximumQueueSize()  {
  return 0;
}

/**
 * Set the maximum number of {@link RASIEvent RASIEvents} which the handler will
 * queue before writing.
 * <p>
 * This handler does not queue events. This method is a no-operation
 * <p>
 * @param size The maximum queue size. Input is ignored.
 */
public void setMaximumQueueSize(int size) throws IllegalStateException  {
  return;
}

/**
 * Return the amount of time (in milliseconds) that this handler will wait before
 * retrying a failed write
 * <p>
 * This handler does not retry or queue failed writes. If a write operation fails,
 * the event is simply discarded.
 * <p>
 * @return The retry interval. Zero is always returned.
 */
public int getRetryInterval()  {
  return 0;
}

/**
 * Set the amount of time (in milliseconds) that this handler will wait before
 * retrying a failed write.
 * <p>
 * This handler does not queue or retry failed writes. This method is a
 * no-operation.
 * <p>
 * @param interval The retry interval. Input is ignored.
 */
public void setRetryInterval(int interval)  {
  return;
}

/**
 * Return the current number of {@link RASIEvent RASIEvents} in the handler's
 * queue.
 * <p>
 * This handler does not queue events. Zero is always returned.
 * <p>
 * @return The current queue size. Zero is always returned.
 */
public int getQueueSize()  {
  return 0;
}

/**
 * Add a RASIFormatter to the set of formatters which are currently registered to
 * this handler. The specified formatter
```

```
 * must be fully configured. Specifically, the formatter must be configured with
 * the set of {@link RASIEvent} classes which it knows how to format.
 * <p>
 * @param formatter The event formatter. Null is tolerated. If the specified
 * formatter supports formatting an event class which already has an
 * associated formatter, the existing formatter is replaced with this one.
 **/
public void addFormatter(RASIFormatter formatter)  {
  if (formatter != null) {
    Enumeration e = formatter.getEventClasses();
    while (e.hasMoreElements()) {
      String name = (String) e.nextElement();
      ivFormatters.put(name, formatter);
    }
  }
}

/**
 * Remove a RASIFormatter from the set of formatters currently registered
 * with this handler.
 * <p>
 * @param formatter The event formatter. If the specified formatter is null
 * or is not registered, this method does nothing.
 **/
public void removeFormatter(RASIFormatter formatter)  {
  if (formatter != null) {
    Enumeration e = formatter.getEventClasses();
    while (e.hasMoreElements())  {
      String name = (String) e.nextElement();
      ivFormatters.remove(name);
    }
  }
}

/**
 * Return an enumeration over the set of RASIFormatters currently registered
 * with this handler.
 * <p>
 * @return An Enumeration over the set of registered formatters. If no
 * formatters are currently registered, the Enumeration is empty.
 **/
public Enumeration getFormatters()  {
  return ivFormatters.elements();
}

/**
 * Close the stream to which this handler is currently writing its entries,
 * if the stream is currently open.
 **/
public void closeDevice()  {
  synchronized(ivDeviceLock) {
    if (ivWriter == null)
      return;
    ivWriter.flush();
    ivWriter.close();
    ivWriter = null;
  }
}

/**
 * Stop the handler, closing the stream to which this handler is currently
 * writing its entries
 * <p>
 * This method must be called when a handler is no longer needed. Be careful
 * not to call this method if other loggers may still be using this handler.
 **/
public void stop()  {
```

```
  /** This handler does not have any queues to flush or preprocessing to do.
   * Simply call
   * closeDevice().
   */
  closeDevice();
}

/**
 * Asynchronously process a RAS event passed from a logger to this handler.
 * <p>
 * WebSphere Application Server loggers always operate synchronously. It is
 * expected that no events will be delivered via this method. This
 * handler also only supports synchronous operations. If events are
 * delivered via this method, simply process them synchronously
 * <p>
 * @param event A RAS event. Null is tolerated
 */
public void logEvent(RASIEvent event)  {
  writeEvent(event);
}

/**
 * Synchronously process a RAS event passed from a logger to this handler.
 * <p>
 * WebSphere Application Server loggers always operate synchronously. It is
 * expected that all events will be delivered via this method.  This handler
 * also only supports synchronous operations.
 * <p>
 * @param event A RAS event. Null is tolerated
 */
public void writeEvent(RASIEvent event)  {
  if (event == null)
    return;
  synchronized(ivDeviceLock) {
    if (ivWriter == null)
      return;
    RASIFormatter formatter = findFormatter(event);
    if (formatter != null) {
      String msg = formatter.format(event);
      ivWriter.println(msg);
      // If an error occurs, simply set the boolean that caller can check
      if (ivWriter.checkError())
        errorHasOccurred = true;
    }
  }
}

////////////////////////////////////////////////////////
//
// Methods introduced by this implementation
//
////////////////////////////////////////////////////////

/**
 * Return the fully-qualified, normalized name of the file which this handler
 *  is currently configured to write events to.
 * <p>
 * @return The fully-qualified, normalized name of the output file.
 */
public String getFileName()  {
  return ivFqFileName;
}

/**
 * Set this handler to write to a file other than the file it is currently
 * writing to.
 * <p>
```

```
 * The current stream that the handler is writing to is closed. A new stream
 * is opened over the specified file.
 * <p>
 * @param name name of the file. May not be null. A fully-qualified file name
 * is recommended.
 * @exception An exception is thrown if the specified name is null, the file
 * cannot be created or some other error occurs. If an exception is thrown,
 * the handlers state is indeterminate.
 */
  public void setFileName(String name) throws Exception  {
  if (name == null)
    throw new Exception("Null passed for name");
  synchronized(ivDeviceLock) {
    closeDevice();
    File x = new File(name);
    ivFqFileName = x.getAbsolutePath();
    openDevice();
  }
}

/**
 * Open a stream over the file to which this handler will write formatted log
 * entries. The stream will always be opened in append mode.
 * <p>
 * If a stream is already open over the file, the current stream is closed.
 * If an error occurs during this operation, the errorHasOccurred boolean is
 * set to true and a plain text error message is written to System.err along
 * with the exception stack trace, if any. If the operation is successful,
 * the errorHasOccurred boolean is set to false.
 * <p>
 */
public void openDevice()  {
  synchronized(ivDeviceLock) {
    try  {
      closeDevice();
      errorHasOccurred = false;
      // The file name may have been changed.Create the directory for the file
      // if it doesn't already exist.
      File x = new File(ivFqFileName);
      String dir = x.getParent();
      File dirs = new File(dir);
      if (fileExists(dirs) == false) {
        boolean result = makeDirectories(dirs);
        if (result == false) {
          errorHasOccurred = true;
          return;
        }
      }
      // Open a file output stream over the file in append mode. Wrap the
      // FileOutputStream in an OutputStreamWriter. Finally wrap the
      // OutputStreamWriter in a BufferedPrintWriter with line flushing
      // enabled.
      FileOutputStream fos = createFileOutputStream(ivFqFileName, true);
      OutputStreamWriter osw = new OutputStreamWriter(fos);
      ivWriter = new PrintWriter(new BufferedWriter(osw), true);
    }
    catch (Throwable t) {
      // not much we can do here except set the error boolean.
      errorHasOccurred = true;
      System.err.println("Error occurred in openDevice() for handler "+ivName);
      t.printStackTrace();
    }
  }
}

/**
 * Return a reference to the formatter associated with the specified event
```

```
 * class. If the specified event class is not registered, the superclasses
 * of the event class will be checked for a registered formatter.
 * <p>
 * @param event A RAS event. Must not be null.
 * @return formatter The formatter associated with the specified event class.
 * Null is returned if the event class is not registered.
 **/
private RASIFormatter findFormatter(RASIEvent event)  {
  Class eventClass = event.getClass();
  RASIFormatter formatter = null;

  while (eventClass != null) {
    String className = eventClass.getName();
    if (ivFormatters.containsKey(className)) {
      return (RASIFormatter) ivFormatters.get(className);
    }
    else
      eventClass = eventClass.getSuperclass();
  }
  return null;
}

/**
 * A worker method that wraps the creation of a FileOutputStream in a
 * doPrivileged block.
 * <p>
 * @param fileName the name of the file to create the stream over.
 * @param append a boolean, when true indicates the file should be opened
 * in append mode
 * @ return the FileOutputStream.
 * @exception SecurityException A security violation has occurred.
 * This class is not authorized to access the specified file.
 * @exception PrivilegedActionException a checked exception was thrown
 * in the course of running the privileged action. The checked exception
 * is contained within the PrivilegedActionException. Most likely the wrapped
 * exception is a FileNotFound.
 **/
 private FileOutputStream createFileOutputStream(String fileName, boolean append)
throws PrivilegedActionException
 {
   final String tempFileName = fileName;
   final boolean tempAppend = append;
   FileOutputStream fs = (FileOutputStream) AccessController.doPrivileged(
     new PrivilegedExceptionAction() {
       public Object run() throws IOException {
         return new FileOutputStream(tempFileName, tempAppend);
       }
     }
   );
   return fs;
}

/**
 * A worker method that wraps the check for the existence of a file in a
 * doPrivileged block.
 * <p>
 * @param fileToCheck a <code>File</code> object whose abstract
 * pathname corresponds to the physical file whose existence is to be checked.
 * @return true if and only if the file exists. Otherwise false.
 * @exception SecurityException A security violation has occurred. This class
 * is not authorized to access the specified file.
 **/
 private boolean fileExists(File fileToCheck) throws SecurityException
 {
   final File tempFileToCheck = fileToCheck;
   Boolean exists = (Boolean) AccessController.doPrivileged(
     new PrivilegedAction() {
```

```
          public Object run() {
            return new Boolean(tempFileToCheck.exists());
          }
        }
      }
    );
    return exists.booleanValue();
  }

  /**
   * A worker method that wraps the creation of directories in a
   * doPrivileged block.
   * <p>
   * @param dirToMake a non-null <code>File</code> object
   * whose abstract pathname represents the fully qualified directory to
   * create.
   * @return true is returned if and only if all necessary directories were
   * created. Otherwise false is returned.
   * @exception SecurityException A security violation has occurred. This
   * class is not authorized to access at leas one of the specified
   * directories.
   **/
  private boolean makeDirectories(File dirToMake) throws SecurityException
  {
    final File tempDirToMake = dirToMake;
    Boolean result = (Boolean) AccessController.doPrivileged(
      new PrivilegedAction() {
        public Object run() {
          return new Boolean(tempDirToMake.mkdirs());
        }
      }
    );
    return result.booleanValue();
  }

}
```

Example: user written formatter: The following is a very simple sample of a
Formatter class. This class is functional, but is intended solely to demonstrate
concepts. For simplicity and clarity, much code (including appropriate boundary
condition checking logic) has been ignored. This sample is not intended to be an
example of good programming practice.

```
package com.ibm.ws.ras.test.user;
import com.ibm.ras.*;
import java.text.*;
import java.util.*;

/**
 * The <code>SimpleFormatter</code> implements the
 * RASIFormatter interface.
 * It is a simple implementation used for demonstration purposes only.
 * It does not do any advanced formatting, it simply formats the message
 * and parameters in an event.
 * It does not include the timestamp in the formatted result, for example.
 */
public class SimpleFormatter implements RASIFormatter
{

  /**
   * The name of the formatter
   */
  private String ivName = "";

  /**
   * A vector containing the event classes this Formatter knows how
   * to process.
   **/
```

```
private Vector ivEventClasses = new Vector();

/**
 * Create a <code>SimpleFormatter</code>.
 **/
public SimpleFormatter(String name)  {
  setName(name);
}

/////////////////////////////////////////////////////
//
// Methods required by the RASIObject Interface
//
/////////////////////////////////////////////////////

/**
 * Return this objects configuration as a set of Properties in a
 * Hashtable.
 * <p>
 * This formatter does not support properties-based configuration.
 * Therefore a call to this method always returns null
 * @return null is always returned.
 */
public Hashtable getConfig()  {
  return null;
}

/**
 * Set this objects configuration from the properties in the specified
 * Hashtable.
 * <p>
 * This formatter does not support properties-based configuration. This method
 * is a no-operation.
 * @param hashTable a Hashtable containing the properties. Input is ignored.
 */
public void setConfig(Hashtable ht)  {
  return;
}

/**
 * Return the name by which this formatter is known.
 * <p>
 * @return a String containing the name of this object, or an empty string
 * ("") if the name has not been set.
 */
public String getName()    {
  return ivName;
}

/**
 * Set the name by which this formatter is known.  If the specified name is
 * <code>null</code>, the current name is not changed.
 * <p>
 * @param name The new name for this object. Null is tolerated.
 */
public void setName(String name)    {
  if (name != null)
     ivName = name;
}

/**
 * Return the description field of this formatter.
 *<p>
 * This formatter does not use a description field. An empty String is always
 * returned.
 *<p>
 * @return an empty String.
```

```
 */
public String getDescription()  {
  return "";
}


/**
 * Set the description field for this formatter.
 *<p>
 * This formatter does not use a description field. Input is ignored and
 * this method does nothing.
 *<p>
 * @param desc The description of this object. Input is ignored.
 */
public void setDescription(String desc)  {
  return;
}


/**
 * Return the name of the {@link com.ibm.ras.mgr.RASManager RASManager} group
 * with which this formatter is associated.  This method is only used by the
 * RAS Manager.
 *<p>
 * This formatter does not support RASManager configuration. Null is always
 * returned.
 * @return null is always returned.
 **/
public String getGroup()  {
  return null;
}

////////////////////////////////////////////////////////
//
// Methods required by the RASIFormatter Interface
//
////////////////////////////////////////////////////////

/**
 * Set a flag that indicates whether this formatter is the default formatter
 * used by {@link com.ibm.ras.RASHandler} objects to format events.
 *<p>
 * Instances of com.ibm.ras.RASHandler are not allowed to be instantiated
 * in the WebSphere Application Server environment.
 * This formatter cannot be the default formatter for handlers of this type.
 * This method does nothing.
 *<p>
 * @param flag input is ignored, since this formatter cannot be the default
 * formatter.
 **/
public void setDefault(boolean flag)  {
  return;
}

/**
 * Return a boolean that indicates whether or not this is the default
 * formatter used by a {@link com.ibm.ras.RASHandler} to format the
 * RAS events.
 *<p>
 * com.ibm.ras.RASHandlers will never be instantiated in a WebSphere
 * Application Server environment so this method always returns
 * false.
 *<p>
 * @return false is always returned.
 **/
public boolean isDefault()  {
  return false;
}
```

```
/**
 * Add the name of a {@link com.ibm.ras.RASIEvent} class to the list of
 * classes which this formatter can process.
 * If the specified class name is null or it is already registered,
 * this method does nothing.
 *<p>
 * @param name The event class name. Null is tolerated.
 **/
public void addEventClass(String name)  {
  if ((name != null) && (! ivEventClasses.contains(name)))
    ivEventClasses.addElement(name);
}

/**
 * Remove the name of a {@link com.ibm.ras.RASIEvent} class from the
 * list of classes which this formatter can process. If the specified
 * class name is null or is not registered, this method does nothing.
 *<p>
 * @param name The event class name.
 **/
public void removeEventClass(String name)  {
  if ((name != null) && (ivEventClasses.contains(name)))
    ivEventClasses.removeElement(name);
}

/**
 * Return an enumeration over the set of names of
 * {@link com.ibm.ras.RASIEvent} classes which this formatter can
 * process.
 *<p>
 * @return An enumeration of RAS event class names. If no event
 * classes are registered, the enumeration is empty.
 **/
public Enumeration getEventClasses() {
  return ivEventClasses.elements();
}

/**
 * Format the specified {link com.ibm.ras.RASIEvent} object and
 * return a String containing the formatted output.
 *<p>
 * @param event The event to format. Null is tolerated.
 * @return The formatted event contents. Null may be returned.
 **/
public String format(RASIEvent event)  {
  if (event == null)
    return null;
  if (event.isMessageEvent())
    return formatMessage(event);
  else
    return formatTrace(event);
}

/**
 * Format a message event.
 *<p>
 * If a message key is used and that key is not found in any
 * message file, the message text becomes an error message indicating
 * that the key was not found.
 *<p>
 * @param event The event to format.
 * @return The formatted event.
 **/
public String formatMessage(RASIEvent event)  {
  String messageKey = "null";
  try {
    messageKey = event.getText();
```

```
      String[] messageInserts = event.getParameters();
      if (event instanceof com.ibm.ras.RASMessageEvent) {
        // RASMessageEvents usually contain localizable messages.
        RASMessageEvent rme = (RASMessageEvent)event;
        String bundleName = rme.getMessageFile();
        if (bundleName != null) {
          // Not a text message, get localized message and return
          ResourceBundle bundle = ResourceBundle.getBundle(bundleName,
          Locale.getDefault());
          String localizedKey =  bundle.getString(messageKey);
          return MessageFormat.format(localizedKey, messageInserts);
        }
        else {
          // Text message
          for (int i=0; i<messageInserts.length; ++i)
          {messageKey = messageKey + " " + messageInserts[i];
          }
          return messageKey;
        }
      }
      else {
        // A User defined type. Append paramaters to key and return
        for (int i=0; i<messageInserts.length; ++i){
          messageKey = messageKey + " " + messageInserts[i];
        }
        return messageKey;
      }
    }
    catch (Throwable t) {
      t.printStackTrace();
      return "SimpleFormattter: Error while formatting message "+messageKey;
    }
  }

  /**
   * Format a trace event.
   *<p>
   * Append the parameters (in order of specification) to the text
   * message in the trace event object.
   *<p>
   * @param event The event to format.
   * @return The formatted event.
   */
  private String formatTrace(RASIEvent event)  {
    String text = "null";
    try {
      text = event.getText();
      String[] parms = event.getParameters();
      if (parms != null) {
        for (int i=0; i<parms.length; ++i){
          text = text + " " + parms[i];
        }
      }
      return text;
    }
    catch (Throwable t) {
      t.printStackTrace();
      return "SimpleFormatter: Error while formatting trace "+text;
    }
  }
}
```

Programming model summary: The programming model described in this section builds upon and summarizes some of the concepts already introduced. This section also formalizes usage requirements and restrictions. Use of the WebSphere

Application Server JRas extensions in a manner that does not conform to the following programming guidelines is prohibited.

As described previously, you can use the WebSphere Application Server JRas extensions in three distinct operational modes. The programming models concepts and restrictions apply equally across all modes of operation.

- You must not use implementation classes provided by the standalone JRas logging toolkit directly, unless specifically noted otherwise. Direct usage of those classes is not supported. IBM Support will provide no diagnostic aid or bug fixes relating to direct usage of classes provided by the standalone JRas logging toolkit.

- You must obtain message and trace loggers directly from the Manager class. You cannot directly instantiate loggers.

- There is no provision that allows you to replace the WebSphere Application Server message and trace logger classes.

- You must guarantee that the logger names passed to the Manager are unique, and follow the naming constraints documented below. Once a logger is obtained from the Manager, you must not attempt to change the name of the logger by calling the setName() method.

- Named loggers are idempotent. For any given name, the first call to the Manager results in the Manager creating a logger that is associated with that name. Subsequent calls to the Manager that specify the same name result in a reference to the existing logger being returned.

- The Manager maintains a hierarchical namespace for loggers. It is recommended but not required that a dot-separated, fully qualified class name be used to identify any given logger. Other than dots or periods, logger names cannot contain any punctuation characters, such as asterisk (*), comma(.), equals sign(=), colon(:), or quotes.

- Group names must comply with the same naming restrictions as logger names.

- The loggers returned from the Manager are subclasses of the RASMessageLogger and RASTraceLogger provided by the standalone JRas logging toolkit. You are allowed to call any public method defined by the RASMessageLogger and RASTraceLogger classes. You are not allowed to call any public method introduced by the provided subclasses.

- If you want to operate in either standalone or combined mode, you must provide your own Handler and Formatter subclasses. You are not allowed to use the Handler and Formatter classes provided by the standalone JRas logging toolkit. User written Handlers and Formatters must conform to the documented guidelines.

- Loggers obtained from the Manager come with a WebSphere Application Server handler installed. This handler will write message and trace records to logs defined by the WebSphere Application Server runtime. Manage these logs using the provided systems management interfaces.

- You can programmatically add and remove user-defined Handlers from a logger at any time. Multiple additions and removals of user defined handlers are allowed. You are responsible for creating an instance of the handler to add, configuring the handler by setting the handler's mask value and formatter appropriately, then adding the handler to the logger using the addHandler() method. You are responsible for programmatically updating the masks of user-defined handlers as appropriate.

- You may get a reference to the handler installed within a logger by calling the getHandlers() method on the logger and processing the results. You must not call any methods on the handler obtained in this fashion. You are allowed to

remove the WebSphere Application Server handler from the logger by calling the logger's removeHandler() method, passing in the reference to the WebSphere Application Server handler. Once removed, the WebSphere Application Server handler cannot be re-added to the logger.

- You are allowed to define your own message type. The behavior of user-defined message types and restrictions on their definitions is discussed in Extending the JRas framework.

- You are allowed to define your own message event classes. The usage of user-defined message event classes is discussed in Extending the JRas framework.

- You are allowed to define your own trace types. The behavior of user-defined trace types and restrictions on your definitions is discussed in Extending the JRas framework.

- You are allowed to define your own trace event classes. The usage of user-defined trace event classes is discussed in Extending the JRas framework.

- You must programmatically maintain the bits in the message and trace logger masks that correspond to any user-defined types. If WebSphere Application Server facilities are being used to manage the predefined types, these updates must not modify the state of any of the bits corresponding to those types. If you are assuming ownership responsibility for the predefined types then you can change all bits of the masks.

JRas Messages and Trace event types
This section describes JRas message and trace event types.

Event types

The base message and trace event types defined by the standalone JRas logging toolkit are not the same as the "native" types recognized by the WebSphere Application Server runtime. Instead the basic JRas types are mapped onto the native types. This mapping may vary by platform or edition. The mapping is discussed below.

Platform Message Event Types

The message event types that are recognized and processed by the WebSphere Application Server runtime are defined in the RASIMessageEvent interface provided by the standalone JRas logging toolkit. These message types are mapped onto the native message types as follows.

WebSphere Application Server native type	JRas RASIMessageEvent type
Audit	TYPE_INFO, TYPE_INFORMATION
Warning	TYPE_WARN, TYPE_WARNING
Error	TYPE_ERR, TYPE_ERROR

Platform Trace Event Types

The trace event types recognized and processed by the WebSphere Application Server runtime are defined in the RASITraceEvent interface provided by the standalone JRas logging toolkit. The RASITraceEvent interface provides a rich and overly complex set of types. This interface defines both a simple set of levels, as well as a set of enumerated types.

- For a user who prefers a simple set of levels, RASITraceEvent provides TYPE_LEVEL1, TYPE_LEVEL2, and TYPE_LEVEL3. The implementations provide

support for this set of levels. The levels are hierarchical (that is, enabling level 2 will also enable level 1, enabling level 3 also enables levels 1 and 2).

- For users who prefer a more complex set of values that can be *OR'd* together, `RASITraceEvent` provides `TYPE_API`, `TYPE_CALLBACK`, `TYPE_ENTRY_EXIT`, `TYPE_ERROR_EXC`, `TYPE_MISC_DATA`, `TYPE_OBJ_CREATE`, `TYPE_OBJ_DELETE`, `TYPE_PRIVATE`, `TYPE_PUBLIC`, `TYPE_STATIC`, and `TYPE_SVC`.

The trace event types are mapped onto the native trace types as follows:

Mapping WebSphere Application Server trace types to JRas `RASITraceEvent` "Level" types.

WebSphere Application Server native type	JRas RASITraceEvent level type
`Event`	TYPE_LEVEL1
`EntryExit`	TYPE_LEVEL2
`Debug`	TYPE_LEVEL3

Mapping WebSphere Application Server trace types to JRas `RASITraceEvent` enumerated types.

WebSphere Application Server native type	JRas RASITraceEvent enumerated types
`Event`	TYPE_ERROR_EXC, TYPE_SVC, TYPE_OBJ_CREATE, TYPE_OBJ_DELETE
`EntryExit`	TYPE_ENTRY_EXIT, TYPE_API, TYPE_CALLBACK, TYPE_PRIVATE, TYPE_PUBLIC, TYPE_STATIC
`Debug`	TYPE_MISC_DATA

For simplicity, it is recommended that one or the other of the tracing type methodologies is used consistently throughout the application. For users who decide to use the non-level types, it is further recommended that you choose one type from each category and use those consistently throughout the application to avoid confusion.

Message and Trace parameters

The various message logging and trace method signatures accept parameter types of `Object`, `Object[]` and `Throwable`. WebSphere Application Server will process and format the various parameter types as follows.

Primitives

Primitives, such as `int` and `long` are not recognized as subclasses of `Object` and cannot be directly passed to one of these methods. A primitive value must be transformed to a proper `Object` type (`Integer`, `Long`) before being passed as a parameter.

Object

`toString()` is called on the object and the resulting `String` is displayed. The `toString()` method should be implemented appropriately for any object passed to a message logging or trace method. It is the responsibility of the caller to guarantee that the `toString()` method does not display confidential data such as passwords in clear text, and does not cause infinite recursion.

Object[]

The `Object[]` is provided for the case when more than one parameter is

passed to a message logging or trace method. `toString()` is called on each `Object` in the array. Nested arrays are not handled. (i.e. none of the elements in the `Object` array should be an array).

Throwable

The stack trace of the `Throwable` is retrieved and displayed.

Array of Primitives

An array of primitive (e.g. `byte[]`, `int[]` is recognized as an `Object`, but is treated somewhat as a second cousin of `Object` by Java. In general, arrays of primitives should be avoided, if possible. If arrays of primitives are passed, the results are indeterminate and may change depending on the type of array passed, the API used to pass the array and the release of the product. For consistent results, user code should preprocess and format the primitive array into some type of String form before passing it to the method. If such preprocessing is not performed, the following may result.

*   [B@924586a0b - This is deciphered as "a byte array at location X". This is typically returned when an array is passed as a member of an `Object[]`. It is the result of calling `toString()` on the `byte[]`.
*   Illegal trace argument : array of long. This is typically returned when an array of primitives is passed to a method taking an `Object`.
*   01040703... : the hex representation of an array of bytes. Typically this may be seen when a byte array is passed to a method taking a single `Object`. This behavior is subject to change and should not be relied on.
*   "1" "2" ... : The String representation of the members of an `int[]` formed by converting each element to an Integer and calling `toString` on the Integers. This behavior is subject to change and should not be relied on.
*   [Ljava.lang.Object;@9136fa0b : An array of objects. Typically this is seen when an array containing nested arrays is passed.

Controlling message logging

Writing a message to a WebSphere Application Server log requires that the message type passes three levels of filtering or screening.

1. The message event type must be one of the message event types defined in the `RASIMessageEvent` interface.
2. Logging of that message event type must be enabled by the state of the message logger's mask.
3. The message event type must pass any filtering criteria established by the WebSphere Application Server runtime itself.

When a WebSphere Application Server logger is obtained from the Manager, the initial setting of the mask is to forward all native message event types to the WebSphere Application Server handler. It is possible to control what messages get logged by programmatically setting the state of the message logger's mask.

Some editions of the product allow the user to specify a message filter level for a server process. When such a filter level is set, only messages at the specified severity levels are written to WebSphere Application Server logs. This means that messages types that pass the message logger's mask check may be filtered out by the WebSphere Application Server itself.

Controlling Tracing

Each edition of the product provides a mechanism for enabling or disabling trace. The various editions may support static trace enablement (trace settings are specified before the server is started), dynamic trace enablement (trace settings for a running server process can be dynamically modified) or both.

Writing a trace record to a WebSphere Application Server requires that the trace type passes three levels of filtering or screening.

1. The trace event type must be one of the trace event types defined in the `RASITraceEvent` interface.
2. Logging of that trace event type must be enabled by the state of the trace logger's mask.
3. The trace event type must pass any filtering criteria established by the WebSphere Application Server runtime itself.

When a logger is obtained from the Manager, the initial setting of the mask is to suppress all trace types. The exception to this rule is the case where the WebSphere Application Server runtime supports static trace enablement and a non-default startup trace state for that trace logger has been specified. Unlike message loggers, the WebSphere Application Server may dynamically modify the state of a trace loggers trace mask. WebSphere Application Server will only modify the portion of the trace logger's mask corresponding to the values defined in the `RASITraceEvent` interface. WebSphere Application Server will not modify undefined bits of the mask that may be in use for user defined types.

When the dynamic trace enablement feature available on some platforms is used, the trace state change is reflected both in the Application Server runtime and the trace loggers trace mask. If user code programmatically changes the bits in the trace mask corresponding to the values defined by in the `RASITraceEvent` interface, the trace logger's mask state and the runtime state will become unsynchronized and unexpected results will occur. Therefore, programmatically changing the bits of the mask corresponding to the values defined in the `RASITraceEvent` interface is not allowed.

Instrumenting an application with JRas extensions
To instrument an application using the WebSphere Application Server JRas extensions, perform the following steps:

Steps for this task
1. Determine the mode the extensions will be used in: integrated, standalone or combined.
2. If the extensions will be used in either standalone or combined mode, create the necessary handler and formatter classes.
3. If localized messages will be used by the application, create a resource bundle as described in Creating JRas resource bundles and message files.
4. In the application code, get a reference to the Manager class and create the manager and logger instances as described in Creating JRas manager and logger instances.
5. Insert the appropriate message and trace logging statements in the application as described in Creating JRas manager and logger instances.

Creating JRas resource bundles and message files: The WebSphere Application Server message logger provides the `message()` and `msg()` methods to allow the

user to log localized messages. In addition, it provides the `textMessage()` method for logging of messages that are not localized. Applications can use either or both, as appropriate.

The mechanism for providing localized messages is the Resource Bundle support provided by the Java Development Kit (JDK). If you are not familiar with resource bundles as implemented by the JDK, you can get more information from various texts, or by reading the javadoc for the `java.util.ResourceBundle`, `java.util.ListResourceBundle` and `java.util.PropertyResourceBundle` classes, as well as the `java.text.MessageFormat` class.

The `PropertyResourceBundle` is the preferred mechanism to use. In addition, note that the JRas extensions do not support the extended formatting options such as {1, date} or {0,number, integer} that are provided by the `MessageFormat` class.

You can forward messages that are written to the internal WebSphere Application Server logs to other processes for display. For example, messages displayed on the administrator console, which can be running in a different location than the server process, can be localized using the *late binding* process. Late binding means that WebSphere Application Server does not localize messages when they are logged, but defers localization to the process that displays the message.

To properly localize the message, the displaying process must have access to the resource bundle where the message text is stored. This means that you must package the resource bundle separately from the application, and install it in a location where the viewing process can access it. If you do not want to take these steps, you can use the early binding technique to localize messages as they are logged.

The two techniques are described as follows:

Early binding
: The application must localize the message before logging it. The application looks up the localized text in the resource bundle and formats the message. When formatting is complete, the application logs the message using the `textMessage()` method. Use this technique to package the application's resource bundles with the application.

Late binding
: The application can choose to have the WebSphere Application Server runtime localize the message in the process where it is displayed. Using this technique,the resource bundles are packaged in a standalone `.jar` file, separately from the application. You must then install the resource bundle `.jar` file on every machine in the installation from which an administrator's console or log viewing program might be run. You must install the `.jar` file in a directory that is part of the extensions classpath. In addition, if you forward logs to IBM service, you must also forward the `.jar` file containing the resource bundles.

To create a resource bundle, perform the following steps.

Steps for this task
1. Create a text properties file that lists message keys and the corresponding messages.

   The properties file must have the following characteristics:
   - Each property in the file is terminated with a line-termination character.

- If a line contains only white space, or if the first non-white space character of the line is the symbol # (pound sign) or ! (exclamation mark), the line is ignored. The # and ! characters can therefore be used to put comments into the file.
- Each line in the file, unless it is a comment or consists only of white space, denotes a single property. A backslash (\) is treated as the line-continuation character.
- The syntax for a property file consists of a key, a separator, and an element. Valid separators include the equal sign (=), colon (:), and white space ( ).
- The key consists of all characters on the line from the first non-white space character to the first separator. Separator characters can be included in the key by escaping them with a backslash (\), but doing this is not recommended, because escaping characters is error prone and confusing. It is instead recommended that you use a valid separator character that does not appear in any keys in the properties file.
- White space after the key and separator is ignored until the first non-white space character is encountered. All characters remaining before the line-termination character define the element.

See the Java documentation for the `java.util.Properties` class for a full description of the syntax and construction of properties files.

2. The file can then be translated into localized versions of the file with language-specific file names (for example, a file named `DefaultMessages.properties` can be translated into DefaultMessages_de.properties for German and DefaultMessages_ja.properties for Japanese).

3. When the translated resource bundles are available, write them to a system-managed persistent storage medium.

Resource bundles are then used to convert the messages into the requested national language and locale.

4. When a message logger is obtained from the JRas manager, it can be configured to use a particular resource bundle. Messages logged via the `message()` API will use this resource bundle when message localization is performed.

At run time, the user's locale setting is used to determine the properties file from which to extract the message specified by a message key, thus ensuring that the message is delivered in the correct language.

5. **(Optional)** If the message loggers `msg()` method is called, a resource bundle name must be explicitly provided.

What to do next

The application locates the resource bundle based on the file's location relative to any directory in the classpath. For instance, if the property resource bundle named DefaultMessages.properties is located in the `<baseDir>/<subDir1>/<subDir2>/resources` directory and `<baseDir>` is in the class path, the name `<subdir1>.<subdir2>.resources.DefaultMessage` is passed to the message logger to identify the resource bundle.

Developing JRas resource bundles: Resource bundle sample

You can create resource bundles in several ways. The best and easiest way is to create a properties file that supports a `PropertiesResourceBundle`. This sample shows how to create such a properties file.

For this sample, four localizable messages are provided. The properties file is created and the key-value pairs inserted into it. All the normal properties files conventions and rules apply to this file. In addition, the creator must be aware of other restrictions imposed on the values by the Java `MessageFormat` class. For example, apostrophes must be "escaped" or they will cause a problem. Also avoid use of non-portable characters. WebSphere Application Server does not support usage of extended formatting conventions that the `MessageFormat` class supports, such as {1, date} or {0,number, integer}.

Assume that the base directory for the application that uses this resource bundle is "`baseDir`" and that this directory will be in the classpath. Assume that the properties file is stored in a subdirectory of `baseDir` that is not in the classpath (e.g. `baseDir/subDir1/subDir2/resources`). In order to allow the messages file to be resolved, the name `subDir1.subDir2.resources.DefaultMessage` is used to identify the `PropertyResourceBundle` and is passed to the message logger.

For this sample, the properties file is named `DefaultMessages.properties`.

```
# Contents of DefaultMessages.properties file
MSG_KEY_00=A message with no substitution parameters.
MSG_KEY_01=A message with one substitution parameter: parm1={0}
MSG_KEY_02=A message with two substitution parameters: parm1={0}, parm2 = {1}
MSG_KEY_03=A message with three parameter: parm1={0}, parm2 = {1}, parm3={2}
```

Once the file `DefaultMessages.properties` is created, the file can be sent to a translation center where the localized versions will be generated.

Creating JRas manager and logger instances: You can use the JRas extensions in integrated, standalone, or combined mode. Configuration of the application will vary depending on the mode of operation, but usage of the loggers to log message or trace entries is identical in all modes of operation.

Integrated mode is the default mode of operation. In this mode, message and trace events are sent to the WebSphere Application Server logs. See Setting up for integrated JRas operation for information on configuring for this mode of operation.

In the combined mode, message and trace events are logged to both WebSphere Application Server and user-defined logs. See Setting up for combined JRas operation for more information on configuring for this mode of operation.

In the standalone mode, message and trace events are logged only to user-defined logs. See Setting up for standalone JRas operation for more information on configuring for this mode of operation.

Using the message and trace loggers

Regardless of the mode of operation, the use of message and trace loggers is the same. See Creating JRas resource bundles and message files for more information on using message and trace loggers.

Using a message logger

The message logger is configured to use the `DefaultMessages` resource bundle. Message keys must be passed to the message loggers if the loggers are using the `message()` API.

```
msgLogger.message(RASIMessageEvent.TYPE_WARNING, this, methodName, "MSG_KEY_00");
... msgLogger.message(RASIMessageEvent.TYPE_WARN, this, methodName, "MSG_KEY_01",
    "some string");
```

If message loggers use the msg() API, you can specify a new resource bundle name.

```
msgLogger.msg(RASIMessageEvent.TYPE_ERR, this, methodName, "ALT_MSG_KEY_00",
"alternateMessageFile");
```

You can also log a text message. If you are using the textMessage API, no message formatting is done.

```
msgLogger.textMessage(RASIMessageEvent.TYPE_INFO, this, methodName, "String and Integer",
"A String", new Integer(5));
```

Using a trace logger

Since trace is normally disabled, trace methods should be guarded for performance reasons.

```
private void methodX(int x, String y, Foo z)
{
   /** trace an entry point. Use the guard to make sure tracing is enabled.
    * Do this checking before we waste cycles gathering parameters to
    * be traced.
    **/
   if (trcLogger.isLoggable(RASITraceEvent.TYPE_ENTRY_EXIT) {
       // since I want to trace 3 parameters, package them up in an Object[]
       Object[] parms = {new Integer(x), y, z};
       trcLogger.entry(RASITraceEvent.TYPE_ENTRY_EXIT, this, "methodX", parms);
   }
... logic
  // a debug or verbose trace point
  if (trcLogger.isLoggable(RASITraceEvent.TYPE_MISC_DATA) {
       trcLogger.trace(RASITraceEvent.TYPE_MISC_DATA, this, "methodX" "reached here");
  }
  ...
  /** Another classification of trace event. Here an important state change has
   * been detected, so a different trace type is used.
   */
  if (trcLogger.isLoggable(RASITraceEvent.TYPE_SVC) {
    trcLogger.trace(RASITraceEvent.TYPE_SVC, this, "methodX", "an important event");
  }
 ...
  // ready to exit method, trace. No return value to trace
   if (trcLogger.isLoggable(RASITraceEvent.TYPE_ENTRY_EXIT)) {
       trcLogger.exit(RASITraceEvent.TYPE_ENTRY_EXIT, this, "methodX");
   }
}
```

Setting up for integrated JRas operation: In the integrated mode of operation, message and trace events are sent to WebSphere Application Server logs. This is the default mode of operation.

Steps for this task

1. Import the requisite JRas extensions classes

   ```
   import com.ibm.ras.*;
   import com.ibm.websphere.ras.*;
   ```

2. Declare logger references.

   ```
   private RASMessageLogger msgLogger = null;
   private RASTraceLogger trcLogger = null;
   ```

3. Obtain a reference to the Manager and create the loggers.

Since loggers are named singletons, you can do this in a variety of places. One logical candidate for enterprise beans is the `ejbCreate()` method. For example, for the enterprise bean named "myTestBean", place the following code in the `ejbCreate()` method.

```
com.ibm.websphere.ras.Manager mgr = com.ibm.websphere.ras.Manager.getManager();
msgLogger = mgr.createRASMessageLogger("Acme", "WidgetCounter", "RasTest",
myTestBean.class.getName());
//Configure the message logger to use the message file created for this application.
msgLogger.setMessageFile("acme.widgets.DefaultMessages");
trcLogger = mgr.createRASTraceLogger("Acme", "Widgets", "RasTest", myTestBean.class.getName())
mgr.addLoggerToGroup(trcLogger, groupName);
```

Setting up for combined JRas operation: In combined mode, messages and trace are logged to both WebSphere Application Server logs and user-defined logs. The following sample assumes that you have written a user defined handler named `SimpleFileHandler` and a user defined formatter named `SimpleFormatter`. It also assumes that you are not using user defined types or events.

Steps for this task

1. Import the requisite JRas extensions classes

   ```
   import com.ibm.ras.*;
   import com.ibm.websphere.ras.*;
   ```

2. Import the user handler and formatter.

   ```
   import com.ibm.ws.ras.test.user.*;
   ```

3. Declare the logger references.

   ```
   private RASMessageLogger msgLogger = null;
     private RASTraceLogger trcLogger = null;
   ```

4. Obtain a reference to the Manager, create the loggers and add the user handlers.

   Since loggers are named singletons, you can obtain a reference to the loggers in a number of places. One logical candidate for enterprise beans is the `ejbCreate()` method. Make sure that multiple instances of the same user handler are not accidentally inserted into the same logger. Your initialization code must handle this. The following sample is a message logger sample. The procedure for a trace logger is similar.

   ```
   com.ibm.websphere.ras.Manager mgr = com.ibm.websphere.ras.Manager.getManager();
       msgLogger = mgr.createRASMessageLogger("Acme", "WidgetCounter", "RasTest",
   myTestBean.class.getName());
       /** Configure the message logger to use the message file defined in the
       * ResourceBundle sample.
       /*
       msgLogger.setMessageFile("acme.widgets.DefaultMessages");

     // Create the user handler and formatter. Configure the formatter,
     // then add it to the handler.
     RASIHandler handler = new SimpleFileHandler("myHandler", "FileName");
     RASIFormatter formatter = new SimpleFormatter("simple formatter");
      formatter.addEventClass("com.ibm.ras.RASMessageEvent");
      handler.addFormatter(formatter);

     // Add the Handler to the logger. Add the logger to the list of the
     // handlers listeners, then set the handlers mask, which will update
     // the loggers composite mask appropriately.
     // WARNING - there is an order dependency here that must be followed.
      msgLogger.addHandler(handler);
      handler.addMaskChangeListener(msgLogger);
      handler.setMessageMask(RASIMessageEvent.DEFAULT_MESSAGE_MASK);
   ```

Setting up for standalone JRas operation: In standalone mode, messages and traces are logged only to user-defined logs. The following sample assumes that you have a user-defined handler named `SimpleFileHandler` and a user-defined formatter named `SimpleFormatter`. It is also assumes that no user-defined types or events are being used.

Steps for this task

1. Import the requisite JRas extensions classes

   ```
   import com.ibm.ras.*;
   import com.ibm.websphere.ras.*;
   ```

2. Import the user handler and formatter.

   ```
   import com.ibm.ws.ras.test.user.*;
   ```

3. Declare the logger references.

   ```
   private RASMessageLogger msgLogger = null;
     private RASTraceLogger trcLogger = null;
   ```

4. Obtain a reference to the Manager, create the loggers and add the user handlers.

   Since loggers are named singletons, you can obtain a reference to the loggers in a number of places. One logical candidate for enterprise beans is the `ejbCreate()` method. Make sure that multiple instances of the same user handler are not accidentally inserted into the same logger. Your initialization code must handle this. The following sample is a message logger sample. The procedure for a trace logger is similar.

   ```
   com.ibm.websphere.ras.Manager mgr = com.ibm.websphere.ras.Manager.getManager();
       msgLogger = mgr.createRASMessageLogger("Acme", "WidgetCounter", "RasTest",
   myTestBean.class.getName());
       // Configure the message logger to use the message file defined in the
       // ResourceBundle sample.
       msgLogger.setMessageFile("acme.widgets.DefaultMessages");

       // Get a reference to the Handler and remove it from the logger.
       RASIHandler aHandler = null;
       Enumeration enum = msgLogger.getHandlers();
       while (enum.hasMoreElements()) {
           aHandler = (RASIHandler)enum.nextElement();
           if (aHandler instanceof WsHandler)
                msgLogger.removeHandler(wsHandler);
       }

       // Create the user handler and formatter. Configure the formatter, then add
       // it to the handler.
       RASIHandler handler = new SimpleFileHandler("myHandler", "FileName");
       RASIFormatter formatter = new SimpleFormatter("simple formatter");
       formatter.addEventClass("com.ibm.ras.RASMessageEvent");
       handler.addFormatter(formatter);

       // Add the Handler to the logger. Add the logger to the list of the handlers listeners,
       // then set the handlers
       // mask, which will update the loggers composite mask appropriately.
       // WARNING - there is an order dependency here that must be followed.
       msgLogger.addHandler(handler);
       handler.addMaskChangeListener(msgLogger);
       handler.setMessageMask(RASIMessageEvent.DEFAULT_MESSAGE_MASK);
   ```

Working with troubleshooting tools

WebSphere Application Server includes a number of troubleshooting tools that are designed to help you isolate the source of problems. Many of these tools are designed to generate information to be used by IBM Support, and their output might not be understandable by the customer.

This section only discusses tools that are bundled with the WebSphere Application Server product. A wide range of tools which address a variety of problems is available from the WebSphere Application Server Technical Support Web site.

Steps for this task
1. Select the appropriate tool for the task.

   For more information on the capacities of the supplied troubleshooting tools, see the relevant articles in this section.
2. Run the tool as described in the relevant article.
3. Contact IBM Support for assistance in deciphering the output of the tool.

Collector Tool

The Collector Tool gathers information about your WebSphere Application Server installation and packages it in a `.jar` file that can be sent to IBM Customer Support to assist in problem determination and analysis. The information includes logs, property files, configuration files, operating system and Java data, and prerequisite software presence and levels.

There are two phases to using the Collector tool. The first phase is to execute the Collector program on your WebSphere Application Server. The second phase is the analysis of the Collector program output `.jar` file by IBM Customer Support.

The Collector program is designed to run to completion despite errors such as files or commands not found, in order to collect as much data as possible.

Running the Collector Tool

The Collector Tool gathers information about your WebSphere Application Server installation and packages it in a `.jar` file that can be sent to IBM Customer Support to assist in problem determination and analysis. The information includes logs, property files, configuration files, operating system and Java data, and prerequisite software presence and levels.

The Collector program is designed to run to completion despite errors such as files or commands not found, in order to collect as much data as possible.

Steps for this task
1. Log on to the system as **root** or **Administrator**.
2. Ensure that Java 1.2.2 or higher is available in the path.

   The Collector program needs Java in order to run, and also collects data about the Java Development Kit (JDK) in which it is running. If multiple JDKs are running on this system, ensure that the JDK being used by WebSphere Application Server is the one in the path for the Collector program. If the JDK being used by the WebSphere Application Server is not available, putting another JDK in the path allows you to collect all the data except Java information.
3. Ensure that the necessary information is in the path being used by the Collector program.

a. If it is a Windows system, `regedit` must be in the path.
b. If it is a UNIX system, the path must contain:
   - /bin
   - /sbin
   - /usr/bin
   - /usr/sbin
4. Create a work directory in which the Collector program is invoked.
5. Make the work directory the current directory.

   The Collector program writes its output `.jar` file to the current directory. It also creates and deletes a number of temporary files in the current directory. Creating a work directory to run the Collector program avoids name collisions and makes cleanup easier. You cannot run the Collector tool in a directory under the WebSphere Application Server installation directory.
6. Run the Collector program by entering the command: `<strong>collector -ServerName <server_name> </strong>` from the command line, where `<server_name>` is the name of the server from which to gather data.

   Note:

   Make sure the path is set up correctly. For Windows, `<WebSphere_home>\bin` must be in the path. For Unix, `<WebSphere_home>/bin` must be in the path.

   The WebSphere Application Server installation directory is determined from the what is specified in "setupcmdline".

   Alternatively, you can fully qualify the path to the collector command. For example, in a Windows default installation, one could enter from the command line: `c:\WebSphere\AppServer\bin\collector.bat.`.

Results

The Collector program creates a log file, `Collector.log`, and an output `.jar` file in the current directory.

The `.jar` file name is based on the hostname and package of the server on which the Collector tool was run, in the format: `<hostname>-<ND|Base>-WASenv.jar`. For example, if the Collector tool were run on the server "ws-laceweb" with a Network Deployment installation, the filename would be "ws-laceweb-ND-WASenv.jar".

The log file is one of the files collected in `<hostname>-<ND|Base>-WASenv.jar`.

What to do next

Send the `<hostname>-<ND|Base>-WASenv.jar` file to IBM Customer Support for analysis.

Analyzing Collector Tool output

The second step in using the Collector tool is to analyze the output. The preferred method of analyzing this output is to send it to IBM Customer Support for analysis, but this article details the contents of the Collector tool output to help you perform your own analysis if necessary.

Although you can view the files contained in the Collector output file without extracting them, it is easier to extract the contents and view them individually. To extract the files, use one of the following commands:

- `jar -xvf <WASenv.jar>`

- `unzip <WASenv.jar>`

Where *Wasenv.jar* is the name of the output `.jar` file created when you ran the Collector tool.

This file contains:
- A Collector execution log file, `Collector.log`
- Copies of stored WebSphere Application Server files and their full paths
- A directory of operating system information in the `OS` directory
- A directory of Java information in the `Java` directory
- A directory of WebSphere Application Server information in the `WAS` directory
- A directory of Collector shell script and batch file execution information for debugging purposes in the `debug` directory
- If MQ is installed, a directory of MQ information in the `MQ` directory
- A jar file manifest.

Tips and suggestions
- Unzip the output `.jar` file to an empty directory for easy access to the gathered files and simplified cleanup.
- Check `Collector.log` for errors.
  - Note that some errors may be normal or expected, as when the Collector tries to gather files or directories that do not exist for your specific installation.
  - A non-zero return code means that the command the tool is trying to execute does not exist. This may be normal, but if it occurs repeatedly it can indicate a problem.
- On Unix systems, the file `OS/commands` has the location of all commands used. If you are missing command output, check this file to see if the command was found.
- On Unix systems, the Collector runs some shell scripts. The shell script output is saved in files in the `OS` directory, while the corresponding debug information is saved in the debug directory. If the output of a shell script is missing, check the corresponding file in the `debug` directory.
- The WebSphere Application Server installation directory is determined from the what is specified in the `PATH` statement. For example, if both the Base Application Server and the Deployment Manager are installed on the same machine, the tool will run against whichever `bin` directory it finds first in the `PATH`. It is recommended that you fully qualify the `PATH` to the Collector tool.
- On Windows systems, the `OS` directory contains a file named `installed.out`. This file contains a list of programs found in the Add/Remove Programs list. This same information is contained in the file `Desktop\My Computer\Control Panel\Add/Remove Programs\Install/Uninstall`.

First Failure Data Capture tool

The First Failure Data Capture tool preserves the information generated from a processing failure and returns control to the affected engines. The captured data is saved in a log file for use in analyzing the problem.

The First Failure Data Capture tool is intended primarily for use by IBM Service. It runs as part of the IBM WebSphere Application Server, and you cannot start or stop it. It is recommended that you not attempt to configure the First Failure Data

Capture tool. If you experience conditions requiring you to contact IBM Service, your IBM Service representative will assist you in reading and analyzing the First Failure Data Capture log.

The First Failure Data Capture tool does not affect the performance of the IBM WebSphere Application Server.

Log Analyzer

The Log Analyzer takes one or more service or activity logs, merges all of the data, and displays the entries. Based on its symptom database, the tool analyzes and interprets the event or error conditions in the log entries to help you diagnose problems. Log Analyzer has a special feature enabling it to download the latest symptom database from the IBM Web site.

To download the latest updates to the symptom database, use the **File -> Update Database -> WebSphere Application Server Symptom Database** option for WebSphere Application Server, or **WebSphere Application Server Network Deployment Symptom Database** option for WebSphere Application Server Network Deployment in the Log Analyzer interface.

About the service or activity log

The application server creates the service or activity log file from the activity of the various WebSphere Application Server components. Log Analyzer is used to view the service or activity log file. Log Analyzer can merge service or activity log files into one log file. The service or activity log file is a binary file located at: *installation_directory*/logs/activity.log.

The service or activity log cannot be viewed using a text editor. The Log Analyzer tool is designed for viewing this file.

Viewing a service or activity log file in the absence of a graphical interface

The Log Analyzer cannot be used to view remote files. If the operating system on which you are running WebSphere Application Server does not support the use of a graphical interface, then transfer the file in binary mode to the system on which you are running the Java administrative console. Use the Log Analyzer tool there.

In cases in which transferring the file is impractical or inconvenient, an alternate tool named "showlog" is provided for viewing the service or activity log file:

1. Change directory to: `<installation_directory>/bin`.
2. Run the showlog tool with no parameters to display the usage instructions:
   - On Windows systems, run `showlog.bat`.
   - On Unix systems, run `showlog.sh`.

To direct the service or activity log (named `activity.log` contents to stdout, use:

`showlog activity.log`

To dump the service or activity log (named `activity.log` to a text file that can be viewed using a text editor, use:

`showlog activity.log [textFileName]`

Using the Log Analyzer

To view the service or `activity.log` using the Log Analyzer:

Steps for this task

1. Change directory to: *install_dir*/bin.
2. Run the waslogbr script file.

   This file is named:
   - `waslogbr.bat` on Windows systems.
   - `waslogbr` on UNIX systems.

   This script must be run from the *install_dir*/bin directory.

   This starts the Log Analyzer interface.
3. Select **File -> Open**.
4. Navigate to the directory containing the service or activity log file.
5. Select the service or activity log file and click **Open**.
6. To analyze the records, right click on a record in the tree on the left, select **UnitOfWorkView** from the right-click menu, and select **Analyze**.

   Now any records with a green check mark next to them match a record in the symptom database. When you select a check-marked record, you will see an explanation of the problem in the lower-right-hand pane.

Log Analyzer main window: The Log Analyzer takes one or more service or activity logs, merges all the data, and, by default, displays the entries in unit of work (UOW) groupings. It analyzes event and error conditions in the log entries to provide message explanations. The Log Analyzer's main window has the following interface:
- Three window panes
- Status line
- Menu bar
- Pop-up actions

Window panes

The Log Analyzer window has three panes:

Logs pane (left)

> By default, Log Analyzer's Logs pane displays log entries by UOW. It lists all the UOW instances and its associated entries from the logs that you have opened. You may find the UOW grouping useful when you are trying to find related entries in the service or activity log or when you are diagnosing problems across multiple machines. The file name of the first log that you opened is shown in the pane's title bar. There is a root folder and under it, each UOW has a folder icon which you can expand to show all the entries for that UOW. All log entries without any UOW identification are grouped into a single folder in this tree view. The UOW folders are sorted to show the UOW with the latest timestamp at the top of the list. The entries within each UOW are listed in the reverse sequence, that is the first (earliest) entry for that UOW is displayed at the top of the list. If you have merged several logs in the Log Analyzer, all the log entries are merged in timestamp sequence within each UOW folder, as if they all came from the same log.

> Every log entry is assigned an entry number, Rec_*nnnn*, when a log is opened in the Log Analyzer. If more than one file is opened in the Log Analyzer (merged files), the Rec_*nnnn* identification will not be unique because the number is relative to the entry sequence in the original log file

and not to the merged data that the Log Analyzer is displaying. This Rec_*nnnn* appears in the first line (**RecoredId**) in the Records pane.

By default, each entry in this pane is color-coded to help you quickly identify the ones that have high severity errors. The values listed here are the default values, you can configure your own colors.

- Non-selected log entry with background color of:
  - Pink indicates that it has a severity 1 error.
  - Yellow indicates that it has a severity 2 error.
  - White indicates that it has a severity 3 error.
- Selected log entry with background color of:
  - Red indicates that it has a severity 1 error.
  - Green indicates that it has a severity 2 error.
  - Blue indicates that it has a severity 3 error.

These colors are configurable and can be changed in the Log Analyzer's Preferences Log page. See Background color for different error severity levels and for more information on how to do this.

The Log Analyzer can display the log entries in different groupings. Use the Log Analyzer Preferences notebook: Logs page to set the grouping filters.

After the Analyze action has been invoked, each analyzed log entry has the following icons:

- A check icon indicates that the entry has some analysis information in one or more pages in the Analysis pane.
- A cascading plus sign (+) icon indicates that the entry has some analysis information and that it has a reraised or remapped exception. You may want to look at the log entry prior to this one when diagnosing problems.
- A question mark icon indicates that the entry has either a severity 1 or 2 error but no additional analysis information is available for it.
- An "x" icon indicates that the entry has a severity 3 error and it has no analysis information.

Record pane (upper right)

When you select an entry in the Logs pane, you see the entry in the Record pane. The entry's identification is shown in the pane's title bar. Right-click in this Record pane to see the actions that you can perform on the entry. There is a drop down arrow next to Record which allows you to go back to look at the last ten records that you have viewed. The cache for the historical data (10, by default) is set in the Preferences General page.

Note:

- The associated analysis data for these cached records are not shown. To see analysis information for cached data, reselect the entry from the Logs pane.

You can enable/disable line wrap mode for the Record Pane using the Log Analyzer Preferences notebook: Record. To print contents of this pane, select **Record > Print** when the Record pane is in focus.

Analysis pane (lower right)

When the analyze action has been invoked and additional information is available, the information will appear in the Symptom page. If the page tab is grayed out, there is no information in that page. The pages of the Analysis pane are:

Symptom

The Log Analyzer provides a database of information on common events and errors to help you recover from some common errors. As a part of the analyze action, if such information is found in the database for the selected log entry, the information is displayed in this page.

Status line

There is a status line at the bottom of the window showing the status of actions.

Menu bar

The menu bar in the Log Analyzer's main window, has the following selections:

File

Open...

Opens a new log file. You can select either a service or activity log or a previously saved XML file. If you want the Log Analyzer to format a raw log file (by running the showlog command) prior to opening it, name the log file with suffix.log. If the Log Analyzer finds that the .log file contains formatted data, it skips the showlog formatting step.

If you want to merge data from another log, select **Merge with**.

Merge with...

When another log file is already opened in the Log Analyzer, use the **Merge with** action to open subsequent logs. The Log Analyzer merges the data from all the logs that it opens and displays all the entries within timestamp sequence in the UOW folders. The data appears as if they came from one log.

If you want the Log Analyzer to format a raw log file (by running the showlog command) prior to opening it, name the log file with suffix.log. If the Log Analyzer finds that the .log file contains formatted data, it skips the showlog formatting step.

Redisplay logs

To redisplay the logs using the recently set filters.

Save as...

Saves the log as an XML file (or text file). If **analyze action** has been performed, all the Symptom analysis information is also saved. If logs are merged in the Log Analyzer, the saved file contains entries of all the merged logs in the sequence that is shown in the Logs pane.

Note: If the merged logs have different timestamp formats, you should not save the merged information because the Log Analyzer only recognizes a single timestamp format for each file that it opens.

Save

Is only enabled if the first file that you opened is an XML file. It resaves the XML file with all the data that is currently displayed in the Log Analyzer. If **analyze action** has been performed, all the Symptom analysis information is also saved. If logs are merged in the Log Analyzer, the saved file contains entries of all the merged logs in the sequence that is shown in the Logs pane.

Note: If the merged logs have different timestamp formats, you should not save the merged information because the Log Analyzer only recognizes a single timestamp format for each file that it opens.

Print Log...

Prints all the entries that the Log Analyzer is displaying. If logs are merged in the Log Analyzer, the output contains entries of all the merged logs in the sequence that is shown in the Logs pane. If analyze action has been performed, all Symptom analysis information is also printed. To print parts of the log, use **Record > Print**.

Close   Closes the opened log.

Update Database

Updates the symptom database which is used for Symptom analysis. It downloads the latest version of the symptom database from the URL specified in the ivblogbr.properties file.

Preferences...

Lets you configure and change the appearance of the Log Analyzer window and its contents.

Exit   Exits the Log Analyzer and closes its window.

Edit

Copy   Copies the selected text in the Record or Analysis pane to the clipboard. If you have not selected any text, **Copy** does not appear in the menu.

Find   Allows you to find text strings in the focused pane.

View

Logs   Toggles the visibility of the Logs pane.

Record

Toggles the visibility of the Record pane.

Symptom

Toggles the visibility of the Symptom page in the Analysis pane.

Record

All the actions under this menu applies to the focused pane.

To select several entries, hold down the **Ctrl** key when making the selection. When a folder is selected, the action applies to all the entries in that folder.

Analyze

> Retrieves and displays additional documentation on known events and event messages in the Analysis pane (Symptom page). Select the folder(s) and/or entries in the Logs pane, right-click to select the **Analyze** action, or from the menu bar, select **Record > Analyze**.
>
> **Note:** If you invoke Analyze for the root folder, then all the entries in the log that you are viewing will be analyzed. If some analysis information is available for an entry, it will either have a check icon or a cascading plus sign (+) icon next to it in the Logs pane. If the analyze action has already been performed, the selection will be grayed out.

Save to file
> Saves the selected entries in the Logs pane. If folders are selected, all the entries in the folder are saved. Any retrieved analysis information is also saved. If the focused pane is either the Records or Analysis pane, then only that pane's information is saved.

Print
> - If the focused pane is Logs, the action prints the selected folder(s) and/or entries. Any retrieved analysis information for those entries is also printed.
> - If the focused pane is Record, the action prints the entry that is currently in the Record pane. Any retrieved analysis information is not printed.
> - If the focused pane is Analysis, the action prints the Symptom page contents.

Windows
> If you have detached the Symptom page in the Analysis pane into separate windows, all the windows will be listed under this menu and you can select the windows that you want to bring to the foreground.

Help Provides a list of on-line documentation for additional information.

Pop-up actions

In the focused pane, right-click to bring up a list of actions in a pop-up menu. Actions that you cannot perform are grayed out. When a folder is selected in Logs pane, the action applies to all the entries in that folder. To select several folders or entries in the Logs pane, hold down the **Ctrl** key when making the selection.

Log Analyzer find window: The Log Analyzer Find window allows you to look for text strings in the focused pane. For example, if you remember the Unit of Work identification, you can enter that text string in the Find window to quickly locate the Unit of Work folder in the Logs pane.

Log Analyzer Preferences notebook - General: The Log Analyzer Preferences notebook's General page lets you specify the behavior of panes in the Log Analyzer's window:

Show title bars
> Shows the title bars of window and its panes.

Highlight selected pane
> Highlights the pane that is in focus.

Pane history cache size
> Specifies a number of records to save in the cache. The Log Analyzer keeps a history of the (specified number of) records that you have viewed. You can use the drop down list next to Record in the pane's title bar to see these cached entries.
>
> **Note:** The associated analysis data for these records are not saved. To see analysis information, reselect the entry from the Logs pane.

Show logo at startup
> Shows the logo when you start-up the Log Analyzer.

When you are finished, click **OK** to apply your changes and close the Preferences notebook.

Log Analyzer Preferences notebook - Appearance: The Log Analyzer Preferences notebook's Appearance page lets you define the overall appearance of the Log Analyzer. You can select the family of products and its texture schemes that you want the Log Analyzer's window to emulate.

When you are finished, click **OK** to apply your changes and close the Preferences notebook.

Log Analyzer Preferences notebook - Toolbars: The Log Analyzer Preferences notebook's Toolbars page lets you customize the appearance and contents of the toolbar in the Log Analyzer window. You can select whether there is text and/or icon in the toolbar, as well as, the functions that you want in the toolbar.

When you are finished, click **OK** to apply your changes and close the Preferences notebook.

Log Analyzer Preferences notebook - Help: The Log Analyzer Preferences notebook's Help page lets you select the browser that will be used to display online help files.

For Windows, the default Web browser will be used and you do not need to update any settings unless there are problems when bringing up the default browser.

For AIX, HP-UX, and Solaris, you have to update the following settings, especially the browser's full path in the **Browser location** entry.

Help browser
> Select the Web browser you want to use.

Browser location
> Select the location of the browser executable file. This should be correct by default, but if you cannot access help then you may need to explicitly enter the browser location.

When you are finished, click **OK** to apply your changes and close the Preferences notebook.

Log Analyzer Preferences notebook - Proxy: The symptom database included in the Log Analyzer package contains entries for common events and errors. New versions of the symptom database provide additional entries. Download new versions of the database from the IBM FTP site. The URL for the FTP site is located in file: *install_dir*/bin/ivblogbr.properties.

The default setting for the FTP site is:
ftp://ftp.software.ibm.com/software/websphere/info/tools/loganalyzer/symptoms/std/symptomdb.xml

You can update your symptom database in one of two ways:
1. Download a new version from the FTP site, and replace your existing database with the new version. Your database is:
   install_dir/symptoms/std/symptomdb.xml.
2. Use the Log Analyzer graphical user interface (GUI) to update your database by selecting: **File -> Update database -> WebSphere Application Server Symptom Database** (for WebSphere Application Server) or **WebSphere Application Server Network Deployment Symptom Database** for WebSphere Application Server Network Deployment.

Setting the proxy definition

If your organization uses a FTP or SOCKS proxy server, contact your system administrator for the host name and port number of the proxy server.

If you use the Log Analyzer GUI to update the database, you can add a proxy definition to the Proxy Preferences page as described below:
1. Select **File -> Preferences -> Proxy**.
2. Select the appropriate proxy type.
3. Enter the host name and port number of the proxy server on the **Proxy** panel.

If you do not use the Log Analyzer GUI, add the proxy definition to the command that launches Log Analyzer.
- Do the following to add the proxy definition for the FTP proxy server:
  - For Windows:
    1. Modify file: *install_dir*\bin\waslogbr.bat.
    2. Add the following text to the file:
       ```
       %JAVA_HOME%\bin\java -DIVB_HOME=%USERPROFILE%/logbr ^
       ....
       -Dftp.proxyHost=proxy_host -Dftp.proxyPort=port_number ^
       ```
  - For UNIX:
    1. Modify file: *install_dir*/bin/waslogbr.
    2. Add the following text to the file:
       ```
       $JAVA_HOME/bin/java -ms10m -mx255m -DIVB_HOME=$HOME/logbr \
       ....
       -Dftp.proxyHost=proxy_host -Dftp.proxyPort=port_number \
       ```
- Do the following to add the proxy definition for the SOCKS proxy server:
  - For Windows:

1. Modify file: *install_dir*\bin\waslogbr.bat.
2. Add the following text to the file:

```
%JAVA_HOME%\bin\java -DIVB_HOME=%USERPROFILE%/logbr ^
....
-DsocksProxyHost=proxy_host -DsocksProxyPort=port_number ^
```

– For UNIX:
1. Modify file: *install_dir*/bin/waslogbr.
2. Add the following text to the file:

```
$JAVA_HOME/bin/java -ms10m -mx255m -DIVB_HOME=$HOME/logbr \
....
-DsocksProxyHost=proxy_host -DsocksProxyPort=port_number \
```

Log Analyzer Preferences notebook -- Logs: The Logs page of the Log Analyzer Preferences notebook lets you group the entries in the logs by different entry fields for viewing. For example, you can select to group the log entries by TimeStamp or clientHostName when they are displayed in the Logs pane.

Primary sort field
Use this filter to set the first level of grouping when log entries are displayed in the Logs pane. By default, the log entries are grouped by UnitOfWork.

Secondary sort field

Use this filter to set the second level of grouping (that is, within the grouping of the primary sort field) when log entries are displayed in the Logs pane.

All the entries within the grouped folders are always sorted in timestamp sequence with the earliest entry at the top of the list.

Redisplay log file immediately
Select this box to immediately regroup the logs entries (after you have clicked **OK**) based on the new filter settings. The entries in the Logs Pane are redisplayed according to the new grouping. If you want to delay the grouping, then do not select this box and, at a later time, you can use the **File > Redisplay logs...** menu selection to regroup and display the log entries based on the changed filter settings.

When you are finished, click **OK** to apply your changes and close the Preferences notebook.

Log Analyzer Preferences notebook -- Severity: The Log Analyzer Preferences notebook's Severity page lets you change the background colors of log entries that appear in the Logs pane. The colors are useful to help you quickly identify entries that have high severity errors and the entry that you have currently selected.

Use colors to indicate severities
Select this checkbox to color-code the background of log entries and folders. When selected, the radio button selections in this page are enabled.

Background color

For each folder and entry in the Logs pane, there is some text describing the entry. To choose a background color for selected log entry that has a severity 1 error, do the following:
1. Select **Selected node**.

2. Select **Severity 1**.

3. Select the color by clicking on the color Swatches. To use the default setting, click **Restore Default**. To see the results of your change, look at the Preview box.

4. Click **Apply** to save that setting.

Repeat similar steps to change the background color for selected log entries that have severity 2 and 3 errors.

To choose a background color for an unselected log entry that has a severity 1 error, do the following:

1. Select **Unselected node**.

2. Select **Severity 1**.

3. Select the color by clicking on the color Swatches. To use the default setting, click **Restore Default**. To see the results of your change, look at the Preview box.

4. Click **Apply** to save that setting.

Repeat similar steps to change the background color for unselected log entries that have severity 2 and 3 errors.

Sample
You can see the result of you color change prior to applying the change. Look at the nodes shown in the Sample box. For color changes of selected nodes, click on the node in the sample box to see the color change.

When you are finished, click **OK** to apply your changes and close the Preferences notebook.

Log Analyzer Preferences notebook -- Analyzer output: The Log Analyzer Preferences notebook lets you enable line wrap for information that appears in the Analysis pane.

Set line wrap
Select the appropriate checkbox to enable line wrap for the Symptom page that appears in the Analysis pane.

When you are finished, click **OK** to apply your changes and close the Preferences notebook.

Diagnosing and fixing problems: Resources for learning

In addition to this InfoCenter, there are several Web-based resources for researching and resolving problems related to the WebSphere Application Server.

The WebSphere Application Server support page

The official site for providing tools and sharing knowledge about WebSphere Application Server problems is the WebSphere Application Server support page: **http://www.ibm.com/software/webservers/appserv/support.html** . Among the features it provides are:

- A search field for searching the entire support site for documentation and fixes related to a specific exception, error message, or other problem. Use this search function before contacting IBM Support directly.

- *Hints and Tips*, *Technotes*, and *Solutions* links take you to specific problems and resolutions documented by WebSphere Application Server technical support personnel.
- A link *All e-fixes, fixpaks, and tools* provides free WebSphere Application Server maintenance upgrades and problem determination tools.
  - e-fixes are software patches which address specific WebSphere Application Server defects. Selecting a specific defect from the list in the *All e-fixes, fixpaks, and tools* page takes you to a description of what problem the e-fix addresses.
  - Fixpaks are rollups of multiple efixes, tested together and released as a maintenance upgrade to WebSphere Application Server. If you select a fixpak from this page, you are taken to a page describing the target platform, WebSphere Application Server prerequisite level, and other related information. Selecting the *list defects* link on that page displays a list of the e-fixes which the fixpak includes. If you intend to install an e-fix which is part of a fixpak, it is usually better to upgrade to the complete fixpak rather than to just install the individual e-fix.
  - Tools are free programs that help you analyze the configuration, behavior and performance of your WebSphere Application Server installation.

Accessing WebSphere Application Server support page resources

Some resources on the WebSphere Application Server support page are marked with a key icon. To access these resources, you must supply a user ID and password, or to register if do not already have an ID. When registering, you are asked for your contract number, which is supplied as part of a WebSphere Application Server purchase.

WebSphere Developer Domain

The Developer Domains are IBM-supported sites for enabling developers to learn about IBM software products and how to use them. They contain resources such as articles, tutorials, and links to newsgroups and user groups. You can reach the WebSphere Developer Domain at **http://www7b.software.ibm.com/wsdd/** .

Obtaining help from IBM

If you are not able to resolve a WebSphere Application server problem by following the steps described in the Troubleshooting guide, by looking up error messages in the message reference, or looking for related documentation on the online help, contact IBM Technical Support.

Purchase of WebSphere Application Server entitles you to one year of telephone support under the Passport Advantage program. For details on the Passport Advantage program, visit www.lotus.com/services/passport.nsf/WebDocs/Passport_Advantage_Home.

The number for Passport Advantage members to call for WebSphere Application Server support is 1-800-237-5511. Please have the following information available when you call:

- Your Contract or Passport Advantage number.
- Your WebSphere Application Server version and revision level, plus any installed e-fixes.
- Your operating system name and version.
- Your database type and version.

- Basic topology data: how many machines are running how many application servers, and so on.
- Any error or warning messages related to your problem.

The Collector Tool

WebSphere Application Server comes with a built-in utility that collects logs and configuration information into one file, the Collector Tool. IBM Technical Support may ask you to run this tool and submit the output.

Tracing

WebSphere Application Server support engineers might ask you to enable tracing on a particular component of the product to diagnose a difficult problem. For details on how to do this, see Enabling trace.

Consulting

For complex issues such as high availability and integration with legacy systems, education, and help in getting started quickly with the WebSphere product family, consider using IBM consulting services. To learn about these services, browse the Web site http://www-1.ibm.com/services/fullservice.html.