# Problem Determination -- table of contents

- 8: WebSphere Application Server Problem Determination
  - 8.1: Problem Determination vs. Tuning
  - 8.1: Applying E-fixes
  - 8.2: Messages
    - How to View Messages
  - 8.3: Logs
    - Log samples
  - 8.4: Traces
    - Trace Samples
    - Enabling and Reading ORB Trace
    - ORB Request Trace
  - 8.5: Identifying the Problem
    - Plug-in Problems
    - Servlet Redirector Problems
    - Workload Management, Remote Administration and Cloning Problems
    - Installation Problems
  - 8.6: Diagnosing Configuration and Installation Problems
  - 8.7: Using Application Level Facilities
    - ORB-related Minor Codes
  - 8.8: Using Internal Tools
    - Using the Log Analyzer for Advanced Edition
  - 8.9: WebSphere Application Server Threads
  - 8.10: Applying E-fixes

8.11: Pointers to other resources

8.12: Various Problem Determination Topics

8.13: Problem Determination Hints and Tips

8.14: How to Report a Problem to IBM

Using the JRas Message Logging and Trace Facility

Contents
Introduction
Overview of messages and trace
The WebSphere JRas programming model
Naming and managing loggers
Message and trace event types
Using JRas loggers
Creating resource bundles and message files
Creating manager and logger instances
Using loggers
Figures
Tables
Notices

## Using the Performance Monitoring Infrastructure Client Package

Contents
Introduction
PMI organization and implementation
PMI client interfaces
Data organization and hierarchy
PMI interfaces
Using the PMI client interfaces
Figures
Tables

# 8: Problem determination

This section provides information about resources and techniques to help you identifyand respond to problems. You can perform problem determination at different levels within your system.Several resources are available for identifying problems:

- Logs
- Trace files
- Messages
- Tools

In order to identify a problem, it is important to understand both the topology of the system and howyour application fits into this topology. See WebSphere Structure in this section. Consider the following questions:

- Are all the components installed successfully?
- What is your application attempting to do?
- How is your application deployed?
- What technology is used to connect to back-end systems?
- Can you re-create the problem?
- What resources best identify the problem?

Next, choose the diagnostic tasks that can help you identifythe component within WebSphere Application Server or within your application that iscausing the problem. Diagnostic tasks include:

- Determining what tuning parameters to specify
- Identifying error messages
- Locating logs and trace files
- Determining whether system and server classpath settings are set correctly
- Identifying failing product components
- Identifying appropriate tools for a problem
- Understanding how to invoke and use available tools

## WebSphere Structure topology

There are a number of configurable values which relate to the server launch. These parametersare held in the path map, process definitions and the OLT configuration. These parametersprovide important details regarding the server java process launch, including debugging details.

The server information is contained within a server configuration file in the serverconfiguration directory. There can be multiple configurations, one of which must be selected bythe user when performing a launch.

You will see a couple of processes running. Instead of an administrative server, there is anadministration enterprise application which runs on top of the application server. These are running inthe same process. The Web server is running in a separate process with the exception of when theinternal Web server is chosen. However, the latter option is not the common operating mode.

The supported application databases are:

- DB2
- Oracle
- instantDB

WebSphere provides the DataSource interface to connect and access these databases. This providesflexibility and efficiency to the application developer because it does not matter which underlying database you choose.

During installation, the Web server that will interact with the WebSphere Application Server is identified. Depending on which Web server is identified, a different plug-in code is installed. The plug-in communicates via HTTP to the internal HTTP server, which thenroutes the requests to the servlet engine or web container.

The application server also contains two subcomponents, the servlet engine/web container and EJB container. The servlet engine interfaces with the plug-in code to service HTTP requests from a Web browser. The EJB container interfaces with the servlet engine or EJB clients to support access to enterprise beans. Both the servlet engine or container can access the customer application data.

## Interfaces

- **RMI/IIOP**

  This interface is provided by the CORBA component of the IBM-supplied Java 2 SDK which is installed with WebSphere Application Server. This interface allows an application to transparently access Java objects that are located either locally or remotely. This interface is also used for interactions between the administrative server, the administrative client and the application server. The SSL security layer is used when the user activates security.

- **HTTP**

  This interface is the externally defined interface used by Web browsers. The

Web server can either service the HTTP request or pass the request to the application server via the OSE interface.

- **JDBC**

  This interface is defined by Java and allows Java programs to access data within the supported databases.

## Logs

The primary WebSphere Application Server processes produce that can be invaluable when doing problem determination. See a description of each log in the Logs section.

# 8.1: Problem determination vs. tuning

This section describes a summary of the difference between problem determination and tuning. Problem determination and tuning are closely related topics, each having the same outcome: a betterperforming product. You might perceive tuning as a subset of problem determination.

Understanding the difference between problem determination and tuning is important. Knowing when to usetuning and when to use problem determination will save you time.

### Problem determination

Problem determination is the process of determining the source of a problem; for example, a program component,machine failure, telecommunication facilities, user or contractor-installed programs or equipment,environmental failure such as a power loss, or user error.

### Tuning

Tuning is the process of adjusting an application or a system to operate in a more efficient manner in thework environment of a particular installation.

In other words, problem determination fixes functional problems, while tuning alleviates slow processes.

The WebSphere Performance and Tuning Guidedescribes the parameters that should be modified to create an optimum product environment.

- 9.1: WebSphere Application Server Tuning Guide

WAS SE

# 8.10: Applying E-fixes

E-fixes are individual fixes for critical problems. They have been individually tested,but not integration tested and should only be applied if you have a critical problem without a valid workaround. They may be applied to both versions of WebSphere, except where specifically noted. All e-fixes are rolled into the next scheduled FixPack. Each fix has a readme file with installation instructions.

To learn about the fixes made available since the last FixPack, see the FixPacks and E-fixes website.

**Related information...**

- 8: Problem Determination

---

View the PDF file containing this article for easy printing

## 8.2: Messages

When WebSphere Application Server is running, it might issue messages related to any of the following components:

- Administrative GUI
- Administrative Repository
- Administrative Server
- Alarm*
- EJB Container
- Connection Manager*
- Database Manager
- Data Replication Service
- Cache Management
- Install
- J2EE Connector
- IBM Java ORB
- Security Association Server
- Java Server Pages
- Localizable Text
- Migration Tools
- JNDI - Name Services
- Web server Plug-ins and Native code
- Resource Analyzer
- Session and User Profiles
- WebSphere Systems Management Utilities
- Servlet Engine
- Tracing Component
- WebSphere Systems Management Commands
- Request Interceptors
- WebSphere Object Adapter
- WebSphere Persistence
- Client
- WSCP Command Line
- WebSphere Server Runtime

- WebSphere Transactions
- WebSphere Systems Management TASKS
- EJB Workload Management
- XML Configurations
- WebSphere Server Process Launch
- WebSphere Server Validation

To help you diagnose problems and minimize the need to enable trace in any of the above components, view the messages table. You can view the messages in alphabetical order by prefix--> or component-->. All messages are documented with user/system action and explanation.

## By Prefix

| | |
|---|---|
| ADGU | Administrative GUI |
| ADMR | Administrative Repository |
| ADMS | Administrative Server |
| CHKJ | IBM Validation Tool |
| ALRM | Alarm* |
| CNTR | EJB Container |
| CONM | Connection Manager* |
| DBMN | Database Manager |
| DRSW | Data Replication Service |
| DYNA | Cache Management |
| INST | Install |
| J2CA | J2EE Connector |
| JORB | IBM Java ORB |
| JSAS | Security Association Service |

| | |
|---|---|
| JSPG | Java Server Pages |
| LTXT | Localizable Text |
| MIGR | Migration Tools |
| NMSV | JNDI - Name Services |
| PLGN | Webserver Plug-ins and Native code |
| PMON | Resource Analyzer |
| SECJ | WebSphere Security |
| SESN | Session and User Profiles |
| SMTL | WebSphere Systems Management Utilities |
| SRVE | Servlet Engine |
| TRAS | Tracing Component |
| WCMD | WebSphere Systems Management Commands |
| WINT | Request Interceptors |
| WOBA | WebSphere Object Adapter |
| WPRS | WebSphere Persistence |
| WSCL | Client |
| WSCP | WSCP Command Line |
| WSVR | WebSphere Server Runtime |
| WTRN | WebSphere Transactions |
| WTSK | WebSphere Systems Management TASKS |
| WWLM | EJB Work Load Management |
| XMLC | XML Configuration |
| WSPL | WebSphere Server Process |
| CHKW | WebSphere Server Validation |

## By Component

| | |
|---|---|
| Administrative Server | ADMS |
| Administrative GUI | ADGU |
| Administrative Repository | ADMR |
| IBM Validation Tool | CHKJ |
| Alarm* | ALRM |
| Cache Management | DYNA |
| Data Replication Service | DRSW |
| IBM Java ORB | JORB |
| Client | WSCL |
| Connection Manager* | CONM |
| Database Manager | DBMN |
| EJB Container | CNTR |
| EJB Work Load Management | WWLM |
| Install | INST |
| J2EE Connector | J2CA |
| Java Server Pages | JSPG |
| JNDI - Name Services | NMSV |
| Localizable Text | LTXT |
| Migration Tools | PMON |
| Resource Analyzer | MIGR |
| Request Interceptors | WINT |

| | |
|---|---|
| Security Association Service | JSAS |
| Servlet Engine | SRVE |
| Session and User Profiles | SESN |
| Tracing Component | TRAS |
| Webserver Plug-ins and Native code | PLGN |
| WebSphere Object Adapter | WOBA |
| WebSphere Persistence | WPRS |
| WebSphere Security | SECJ |
| WebSphere Server Runtime | WSVR |
| WebSphere Systems Management Commands | WCMD |
| WebSphere Systems Management TASKS | WTSK |
| WebSphere Systems Management Utilities | SMTL |
| WebSphere Transactions | WTRN |
| WSCP Command Line | WSCP |
| XML Configuration | XMLC |
| WebSphere Server Process Launch | WSPL |
| WebSphere Server Validation | CHKW |

## 8.2.1: How to view messages

All messages will show in the shell window from which the application server was started. You canalso have these messages routed to a file by updating the trace service object for a particularapplication server definition. This can be done by using the administrative client web interface.If you select to have these messages routed to a file, the administration client allows you to viewthe contents of the file from the browser.

# 8.3: Logs

WebSphere Application Server provides many error logs to help you diagnoserun-time problems. This section describes these error logs telling you where to find andhow to format the files. The logs are:

- activity.log
- stderr.log
- stdout.log
- Plug-in log
- NT wssetup.log
- UNIX WebSphere.instl
- Serious error log

The tools required to process some of these logs (as well as some of the trace logs) are described in Using Internal Tools. You can also refer to Problem determination hints and tips for additional tips on the use and processing of some of these error logs. If you need to report a problem to IBM, you might need to gather some of these error logs and send them to IBM for diagnosis; for moreinformation, refer to How to report a problem to IBM.

## Activity log for problem determination

The activity log captures events that show a history of WebSphere Application Server's activities. Some of the entries in the log are informational, while others report on systemexceptions, such as returned CORBA exceptions.

When you encounter WebSphere Application Server run-time errors, you will often find it usefulto use Log Analyzer to read the activity log and try to diagnose the problem yourself. When you need assistance from IBM to help you diagnose problems, you will be asked to providethe formatted activity log output to IBM.

### Location of the activity log

There is one activity log for each host machine. The activity.log file resides in the logsdirectory of where the product is installed. All application servers, including the administrativeserver, write error records to this file. The activity.log file is a binary file and cannotbe viewed with an ascii editor. You can view the activity.log file in one of two ways:

- Log Analyzer
- showlog

**NOTE:**The activity.log file should NOT be edited. If sections are deleted from this filethe file will become corrupted.

### How to view the activity.log file with Log Analyzer

1. Change the directory to:
   product_installation_root/bin
2. Run the waslogbr script file, which is called:
   - waslogbr.bat on Windows NT
   - waslogbr.sh on Unix systems

   It needs to be run from the bin directory cited above.
   This will start the Log Analyzer graphical interface.
3. In the interface:
   1. Select File>Open.
   2. Navigate to the directory containing the activity.log file.
   3. Select the activity.log file.
   4. Select Open.

### How to view the acitivity.log file on a remote machine using showlog

If you plan to transfer the activity.log file to a remote machine, you must transer the file usinga tool such as FTP. The file MUST be transferred in binary mode, otherwise the log file could corrupt and will not be readable.

The Log Analyzer cannot be used to view remote files.An alternate tool named showlog can be used instead of Log Analyzer to format the activity.log file for viewing when no GUI display capabilities are available.

showlog.bat or showlog.sh is a script/batch file that can be found in the bin directoryof the WebSphere Application Server installation. Follow these instructions to use showlog:

1. Change directory to:
   product_installation_root/bin
2. Run the showlog tool with no parameters to display the usage instructions:
   - On Windows NT, run showlog.bat

    - On Unix systems, run showlog.sh

Examples:

- To direct the activity log contents to stdout, use the invocation:
  showlog activity.log
- To dump the activity.log to a text file that can be viewed using a text editor, use the invocation:
  showlog activity.log textFileName

### Changing activity.log file size

In the course of using Log Analyzer, you might have to set the maximum activity.log file size.The activity.log file grows to a predetermined size and then wraps. The default size is 1MB. Followthese steps to change the log size:

1. Open the properties file in a text editor:
   product_installation_root/properties/logging.properties
2. For the com.ibm.ws.ras.ActivityLogSize property, specify the value you would like in Kilobytes (KB). If an individual size is entered, the default size is used.
    - Example: To change the log size to 2MB, enter in the line: com.ibm.ws.ras.ActivityLogSize=2048 (do not use spaces)

The size change will take effect at the next server startup.

When making changes to the acitivity log, remember that the activity log uses a lockfile named activityLog.lck, located in the same directory as the activity log, to synchronize acces to the activity log. If you use either showlog or the Log Analyzer, youmust have write access to the /logs/directory. These programs must lock the activity log while making a copy of it. In order to do this, the programsmust be able to create the lock file in this directory which requires write access.

## stderr and stdout logs for problem determination

The stderr and stdout logs capture events presented through two of the three standard I/O streams, or:

- stdin - arguments entered with a command or program
- stdout - output displayed to the user
- stderr - errors thrown by the code

In WebSphere Application Server, the stdout and stderr logs are created for:

- Application servers
- Servlet redirectors

The application server stderr and stdout logs contain application server communication.Output from `System.err.println` and `System.out.println` statements in the servlet code also appear in the application server stdout and stderr logs.

## Plug-in logs for problem determination

The native.log file is created by the plug-in running inthe Web server process. This file are located in the ./logs directory of theWebSphere installation. Different levels of information can be placed in this log.This log contains error and informational messages generated from the Web server plug-in.This information reflects server startup and server status change requests (start/stop/restart).

The default log file mask setting for the plug-in log is **error**.If this log has a file length of zero, no error messages were generated during the server status change requests.

## NT wssetup log

This log is created during the install process. Review this log to ensure the install process was successful. The install process consists of:

- Verifying prerequisites
- Downloading files
- Updating the configuration files for both WebSphere Application Server and the Web server

## UNIX WebSphere.instl

On AIX and Solaris, a native install of WebSphere Application Server generates theWebSphere.instl log that is located in the `/tmp` directory.

Information on the WebSphere Application Server install process on HP is placed in the HP system log, swagent.log, that is located in the`/opt/WebSphere/AppServer/var/adm/sw` directory.

# Serious error log

If a fatal error occurs, the serious error log file may be produced. This log contains the server nameand text that reads "fatal error."

### 8.3.1: Log samples

# 8.4: Traces

Traces are just logs.Traces and logs differ in that you must turn traces on to see output in atrace file. Logs are always enabled and log entries are automatically generated.

Tracing occurs as a single process for the administrative and application servers.

Trace can be enabled for any "trace component" that has registered with the trace system.Typically, a trace component and a Java$^{\text{TM}}$ class have the same range, although itis not required. There are some trace components that do not follow this such as the entireORB component. The ORB component consists of multiple java classes, but registers as a singletrace component. Determining the granularity of a trace component is left up to the descretionof the developer or component. Review the WebSphere Application Server Java package names in the table underIdentifying the Problem.This table includes some of the Java classes that can be traced.

The trace subsystem does not trace user code (such asservlets or EJB components) unless `System.err.println` or `System.out.println` statements are added to the code. Output from the `println` statements appears either in the application server stdout or stderr logs. A Trace/Log API called JRas is also availableto trace servlets and EJB components. User code instrumented with JRas will behave exactly like otherruntime trace.

See the stdout and stderr logs description for more information on stdout and stderr logs.

Beginning with WebSphere Application Server Version 3.0,an object level debugger is provided with the product to trace and debug user code.See the Object Level Tracing and Debugging (OLT and OLD) section for information on object level tracing.

## Enabling Trace

Trace for a server can either be enabled before the server is started startup trace enablement or can beenabled while the server is up and running dynamic trace enablement.

### Startup trace enablement

Enabling an administrative server is different from enabling an application server. For the administrativeserver, you need to edit the admin.config file to set the com.ibm.ejs.sm.adminServer.traceString property.For an application server, you need to launch the administrative GUI and follow these steps:

1. Click on the server.
2. On the righthand pane, select the Services tab.
3. Select Trace Service.
4. Select Edit Properties.
5. Enter the desired trace specification.
6. Select OK.
7. Select Apply.

SeeViewing traces/collecting traces for more information on writing trace specifications.

## Trace and log entry format

WebSphere Application Server supports multiple trace formats which are specifiable by the user.There are three formats:

- Basic
- Advanced
- Loganalyzer

### Basic

Since trace is just another log, both a WebSphere Application Server log entry anda trace entry will have the same format. The following example of a log entry illustratesthe basic format:

Log entry example: **[00.07.11 22:47:12:191 EDT] 53ccc3c5 ActiveEJBCont W Could not create bean table xxx**

The following table includes a description of each of part of the log entry:

| [00.07.11 22:47:12:191 EDT] | 53ccc3c5 | ActiveEJBCont | W | Could not create bean table | xxx |
|---|---|---|---|---|---|
| **TS**: The timestamp in fully qualified date (YYMMDD), Time (Millisecond precision), and Time zone format. | **TID**: The thread ID or the hash code of the thread issuing this message. | **COMPONENT**: The short name of component issuing this message. | **LEVEL**: The level of the message or trace. Possible levels are:<br>● > Entry to a method (debug)<br>● < Exit a method (debug)<br>● **A** Audit<br>● **W** Warning<br>● **X** Error<br>● **E** Event (debug)<br>● **D** Debug (debug)<br>● **T**Terminate (exits process)<br>● **F** Fatal (exits process) | **MESSAGE**: The text of the message. | **ARGUMENTS**: Optional message arguments. |

### Advanced

The following is a sample of the advanced format:

**[01.05.24 15:06:514 CDT] 1014f419 I UOW=1-829:Default Server source=com.ibm.ws.runtime.utils.ResourceBinder org=IBM prod=WebSphereWSVR00491: Binding SampleDataSource as jdbc/SampleDataSource**

The following table includes a description of each of part of the log entry:

| [01.05.24 15:06:514 CDT] | 1014f419 | UOW=1-829:Default Server | I | source=com.ibm.ws.runtime.utils.ResourceBinder | org=IBM | prod=WebSphere | Component (see Note 1) | WSVR00491 | Binding SampleDataSource as jdbc/SampleDataSource |
|---|---|---|---|---|---|---|---|---|---|
| **TS**: The timestamp in fully qualified date (YYMMDD), Time (Millisecond precision), and Time zone format. | **TID**: The thread ID or the hash code of the thread issuing this message. | **CORRELATION ID**: Generated by the runtime. | **LEVEL**: The level of the message or trace. Possible levels are:<br>● > Entry to a method (debug)<br>● < Exit a method (debug)<br>● **A** Audit<br>● **W** Warning<br>● **X** Error<br>● **E** Event (debug)<br>● **D** Debug (debug)<br>● **T**Terminate (exits process)<br>● **F** Fatal (exits process) | **SOURCE**: The name of the component or class issuing the message. | **ORGANIZATION**: The organization who wrote this code. | **PRODUCT**: Name of the product. | **COMPONENT**: Name of the component. | **MESSAGE**: The text of the message. | **ARGUMENTS**: Optional message arguments. |

**Note 1:** Organization, product and component can be set on JRas loggers. For existingWebSphere runtime code, defaults are provided for the organization and product, and do not displaya component.

### Loganalyzer

The loganalyzer format is useful for combining and correlating traces from multiple server processes.Move all the trace files to a directory on a single system. The trace files MUST have been generated in loganalyzer format. Launch the Log Analyzer and use **File**->**Open** to navigate to that directoryand open one of the trace files. Next, use the **File**->**Merge with** and select another trace file.This will merge the contents of the two files in the Log Analyzer display.

#### Useful information when using the advanced and loganalyzer formats

If the startup trace is not enabled, the stdout.log and stderr.log files are always generated in basic format.If startup trace is enabled and sent to a user specified file, then the stdout.log and stderr.log files are generated in basic format and the user-specified file is generated in the format specified by the user.

If startup trace is enabled and sent to stdout.log or stderr.log, then that file is generated in the formatspecified by the user. The ring buffer is always generated in the format specified by the user.

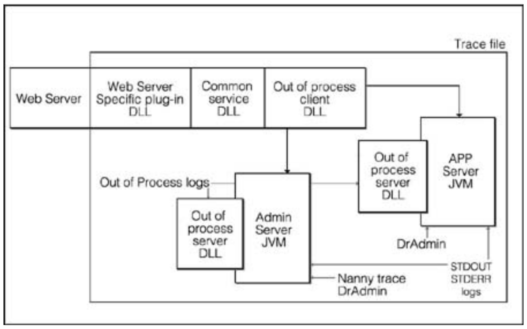
If a persistent file is configured, then the data in the file is generated in the selected format and the messages sent to the shell are always in basic format. If a persistent file is NOT configured, then the messagessent to the shell are generated in the specified format.

## Types of traces

The following are the traces you will find in WebSphere Application Server:

- Trace file
- UNIX Nanny trace
- DrAdmin

View the following graphic for a description of the log and trace points in WebSphere Application Server:



### Trace file

The trace file provides trace entries on the interaction of various WebSphere ApplicationServer components with the administrative server.Use the trace file to identify a problem and to review events preceding the error situation.

**Note:** Always review trace entries prior to the error. Trace entries recorded after the error has occurred represent program recovery and will nothelp with problem determination.
Review the Collecting traces section for additional tracedocumentation.

Trace file samples

### UNIX Nanny trace

On UNIX platforms, the nanny process starts the administrative server.The nanny.maxtries parameter in the admin.config file tells the nanny process how many timesit should attempt to restart the administrative server.

On Windows NT, the nannyservice is part of the IBM WebSphere Administrative Server service that is registered with the operating system.Starting the IBM WebSphere Administrative Server service invokes adminservice.exe. If the service does not start, verify that:

- The service was installed and is available from `Start > Settings > Control Panel > Services`
- The userID under which WebSphere Application Server was installed has service privileges

If the nanny process fails to start the administrative server on UNIX or if the IBM WebSphere Administrative Server service does not start on Windows NT, you can bypass the nanny function and just start the administrative server. Follow these steps to start the administrative server:

1. Go to the <WAS_root>bin/bin directory
2. Invoke adminserver.sh on UNIX or adminserver.bat on Windows NT

   **Note:** Starting the administrative server without using the nanny function means that nothing is monitoring the administrative server. If it fails in thisstate, nothing will restart it.

A nanny trace is only available on UNIX platforms.

On Windows NT, use the Event Viewer to view entries related to the WebSphere nanny service. Followthese steps to view the Event Viewer :

1. Select Start > Programs > Administrative Tools
2. Select Event Viewer
3. View events related to WebSphere Application Server

Nancy trace samples

### DrAdmin trace function

The DrAdmin function generates thread dumps.

On UNIX platforms, the IBM JDK allows users to send signals to force javacore.txt filesto be created in the application server's working directory.The application server continues to run and a sequence of javacore files are created. These filescan help in debugging "loop" or "system hang" problems.

To generate thread dumps similar to the javacore files, especially on a Windows NTplatform, use the DrAdmin function.

A unique DrAdmin port is generated each time an application server starts. To generate a thread dump for that port:

1. View the console messages area or the trace file for message SMTL0018I "DrAdmin available on port."
2. Enter the following command:
   ```
   DrAdmin -serverport <port number> -dumpThreads
   ```
3. Review the stderr log for the thread dump.

**Note:** A specific DrAdmin port may be configured. If the DrAdmin port is set to -1, which is a default, this indicates thata specific port has not been set and a port will automatically be generated.

After installing and starting WebSphere Application Server, you will see DrAdmin entries in the console messages area. These entries appear regardless of the options specified during installation, and have the following format:

> DrAdmin available on port 1,055

DrAdmin entries are also recorded in the trace file.To locate the DrAdmin entry in the trace file:

1. In the <WAS_ROOT>**logs** directory, open the trace file.
2. Go to the bottom of the trace file and then scroll up until you locate the following entry:`DrAdmin available on port xxxx`

### What is DrAdmin?

DrAdmin is a service, provided by each of the servers, to enable and disable tracing. Each time a server starts, DrAdmin registers itself on a different (next available) port number. There are no output messages associated with DrAdmin. The DrAdmin entries in the console messages area are generated to tell users the port number where DrAdmin is listening.

### When to use DrAdmin?

You should always use the administrative console trace facilities to debug a problem. DrAdmin provides useful lightweight access to several runtime functions, as well as provides access to a numberof different processes, including nanny and administrative server processes. Opt to use DrAdmin when the followingsituations occur:

- When input to the administrative console is not accepted
- When the administrative server is in a wait state
- When the administrative server is not responding (e.g., in an infinite loop orhung state)
- When you have to dump the thread stacks in a server
- When the administrative client topology tree disappears

**Note:** DrAdmin is an internal interface that is used to assist users with problem determination. As aninternal interface, it is subject to change at any time, and there is no national language support for it.

### How to use DrAdmin?

The DrAdmin interface is the same on all platforms. Since DrAdmin is another way of turning on a trace, the tracing mechanism is the same as the one used by the administrative console trace facilities.Therefore, whether you are looking at the trace file or a DrAdmin output file,the trace entries will have the same format.
See the DrAdmin samples to learn how to invoke DrAdmin.

Use of the admin.config file for trace output does not apply to the single server product.Instead, server trace options are configured through the server configuration file:

```
INFO_USAGE_LINE_37=(Prepend the trace file with "!" to cause that file to
INFO_USAGE_LINE_38=be truncated when starting the server. Use the values
INFO_USAGE_LINE_39="stdout" or "stderr" to case trace output to be written,
INFO_USAGE_LINE_40=respectively, to standard output or to standard error.)
```

#### DrAdmin Help

DrAdmin has a help file available. You can access the DrAdmin help by using typing on the command linethe "- help" option.

DrAdmin samples

For more information on traces, see file What are messages, logs and traces?

For a quick overview of available traces and tools, see the Problem/Tool/Trace/Log Matrix.

## 8.4.1: Trace samples

Select one of the following traces to view sample output:

- Trace file
- UNIX Nanny trace (available on UNIX platforms only)
- DrAdmin

### Trace file

The following trace fragment is an example of a trace file.Use this trace to debug server startup or shutdown problems:

```
[00.07.17 15:59:57:200 EDT] f0c45c4c AdminServer   A Initializing WebSphere
Administration server[00.07.17 15:59:57:230 EDT] a9d5dc4e DrAdminServer A DrAdmin
available on port 1,038[00.07.17 16:00:22:457 EDT] f0c45c4c SASConfig     A SAS
Property:com.ibm.CORBA.principalName has been updated[00.07.17 16:00:25:191 EDT]
f0c45c4c InitialSetupI A Creating Sample Server Configuration[00.07.17 16:00:29:797
EDT] f0c45c4c JDBCDriverCon A Importing JDBCDriver : Admin DB Driver[00.07.17
16:00:31:209 EDT] f0c45c4c JDBCDriverCon A Installing JDBC Driver: Admin DB Driver on
node db[00.07.17 16:00:31:530 EDT] f0c45c4c JDBCDriverCon X Failed to install JDBC
Driver Admin DB Driver  on node db.OpException
com.ibm.ejs.sm.exception.JDBCDriverAlreadyInstalledException[00.07.17 16:00:31:770
EDT] f0c45c4c DataSourceCon A Importing DataSource : Default DataSource[00.07.17
16:00:32:451 EDT] f0c45c4c NodeConfig    A Importing Node : db[00.07.17 16:00:32:962
EDT] f0c45c4c ApplicationSe A Importing ApplicationServer : Default Server[00.07.17
16:00:33:823 EDT] f0c45c4c ContainerConf A Importing Container : Default
Container[00.07.17 16:00:35:746 EDT] f0c45c4c EJBConfig     A Importing EJB :
HitCount Bean[00.07.17 16:00:37:859 EDT] f0c45c4c EJBConfig     A Importing EJB :
BeenThere Bean[00.07.17 16:00:39:271 EDT] f0c45c4c ServletEngine A Importing
ServletEngine : Default Servlet Engine[00.07.17 16:00:40:843 EDT] f0c45c4c
WebApplicatio A Importing WebApplication : default_app[00.07.17 16:00:44:088 EDT]
f0c45c4c ServletConfig A Importing Servlet : snoop[00.07.17 16:00:44:248 EDT]
f0c45c4c ServletConfig W Updating Servlet : snoop, since it was already
created[00.07.17 16:00:48:604 EDT] f0c45c4c ServletConfig A Importing Servlet :
hello[00.07.17 16:00:48:694 EDT] f0c45c4c ServletConfig W Updating Servlet : hello,
since it was already created[00.07.17 16:00:51:508 EDT] f0c45c4c ServletConfig A
Importing Servlet : ErrorReporter[00.07.17 16:00:51:609 EDT] f0c45c4c ServletConfig W
Updating Servlet : ErrorReporter, since it was already created[00.07.17 16:00:53:982
EDT] f0c45c4c ServletConfig A Importing Servlet : invoker[00.07.17 16:00:54:182 EDT]
f0c45c4c ServletConfig W Updating Servlet : invoker, since it was already
created[00.07.17 16:00:56:586 EDT] f0c45c4c ServletConfig A Importing Servlet :
jsp10[00.07.17 16:00:56:806 EDT] f0c45c4c ServletConfig W Updating Servlet : jsp10,
since it was already created[00.07.17 16:01:02:825 EDT] f0c45c4c WebApplicatio A
Importing WebApplication : admin[00.07.17 16:01:05:428 EDT] f0c45c4c ServletConfig A
Importing Servlet : install[00.07.17 16:01:05:539 EDT] f0c45c4c ServletConfig W
Updating Servlet : install, since it was already created[00.07.17 16:01:07:982 EDT]
f0c45c4c ServletConfig A Importing Servlet : jsp10[00.07.17 16:01:08:092 EDT]
f0c45c4c ServletConfig W Updating Servlet : jsp10, since it was already
created[00.07.17 16:01:14:271 EDT] f0c45c4c ServletConfig A Importing Servlet :
file[00.07.17 16:01:14:361 EDT] f0c45c4c ServletConfig W Updating Servlet : file,
since it was already created[00.07.17 16:01:16:865 EDT] f0c45c4c ServletConfig A
Importing Servlet : invoker[00.07.17 16:01:16:975 EDT] f0c45c4c ServletConfig W
Updating Servlet : invoker, since it was already created[00.07.17 16:01:19:439 EDT]
f0c45c4c ServletConfig A Importing Servlet : ErrorReporter[00.07.17 16:01:19:529 EDT]
f0c45c4c ServletConfig W Updating Servlet : ErrorReporter, since it was already
created
```

### UNIX Nanny trace

The following trace fragment is an example of a nanny trace.Use the nanny trace to monitor administrative server events:

```
[00.07.17 17:05:00:032 EDT] 1fa4cc16 Nanny         > main
"admin.config"[00.07.17 17:05:00:032 EDT] 1fa4cc16 Nanny         > Initial admin
server startup..[00.07.17 17:05:06:231 EDT] 1fa4cc16 Nanny         < Initial
adminserver startup successful..[00.07.17 17:05:06:321 EDT] 1fa50b45 Nanny         >
run : AdminServerMonitorThread[00.07.17 17:05:06:321 EDT] 1fa50b45 Nanny         E
AdminServerMonitorThread: Waiting for process 1719 to terminate.
```

### DrAdmin

NT To invoke DrAdmin:

1. Go to the <WebSphere\AppServer\bin\debug> directory.
2. Copy adminserver.bat to DrAdmin.bat

   UNIX **Note:**On Unix platforms, the adminserver.bat file is adminserver.sh. Copy adminserver.sh to DrAdmin.sh.
3. Replace the following line in the DrAdmin.bat file:
   %JAVA_HOME%\bin\java -mx128m com.ibm.ejs.sm.server.AdminServer -bootFile %WAS_HOME%\bin\admin.config %restart% %1 %2 %3 %4

   with

   %JAVA_HOME%\bin\java com.ibm.ejs.sm.util.debug.DrAdmin %1 %2 %3 %4 %5 %6 %7 %8 %9
4. Save and close the DrAdmin.bat file
5. From a command prompt in the <WebSphere\AppServer\bin\debug> directory, type
   DrAdmin [options]where options are:
   - -help [shows the help message]
   - -serverHost <Server host name> [Specify the host name of the server... defaults to local host]
   - -serverPort <Server port number> [Required... enter the port number where DrAdmin is listening]
   - -setTrace <Trace specification> [Specify any valid traceString, for example, "com.ibm.ejs.sm.*=all=enabled"]
   - -setRingBufferSize <Number of ring buffer entries in k> [Specify the number of trace entries to store in the main memory buffer... the default is 8k]
   - -dumpRingBuffer <Name of file to dump the ring buffer> [Defaults to file name JMONDump.xxxxxxxxxxxx where xxxxxxxxxxxx is a combination time of day and unique PID identifier extension]

     NT **Note:** On Windows NT, if the administrative server is started as a service, the default DrAdmin dump file will be located in the <Winnt\system32> directory.
   - -dumpState <dumpString> [Specify a unique identifier for this dump]
   - -stopServer [Stops the administrative server]
   - -stopNode [Does not apply unless the node is connected to the administrative server]
   - -dumpThreads [Dumps the threads in the server]
   - -testConnection [Determines if the DrAdmin server is running]
   - -retrieveServerNames [Shows names of the server associated with DrAdmin]
   - -retrieveTrace [Retrieves the current trace specification]
   - -retrieveComponents [Retrieves a list of the current active trace components]
   - -dumpConfig [Dumps configuration information to the server standard output]
   - -retrieveConfig [Dumps configuration information to the DrAdmin command line]
   - -list [Lists installed web applications and modules]
   - -long [Lists, in long format, installed web applications and modules]

**Another example of implementing a DrAdmin trace:**

NT On Windows NT:

1. Create a DrAdminRun.bat file that contains the following information:`set CLASSPATH=C:/jdk1.1.7/lib/classes.zip;C:/WebSphere/AppServer/lib/ujc.jar;C:/WebSphere/AppServer/lib/ejs.jar;C:/WebSphere/AppServer/lib/admin.jarecho %CLASSPATH%echoechojava -classpath %CLASSPATH% com.ibm.ejs.sm.util.debug.DrAdmin -serverPort %1 -setTrace %2=%3=%4`
2. Invoke DrAdminRun.bat with the port number and the trace string. Use the port number from DrAdmin entry in the trace file, xxxx. Your input from a command prompt will be: `DrAdminRun xxxx com.ibm.ejs.* all enabled`
3. Start administrative client with the debug option by invoking adminclient.bat from the WAS_ROOT bin directory: `adminclient debug`

UNIX On UNIX platforms:

1. Create a shell script file DrAdminRun that contains the following information:`# modify classpath as appropriate for platform/environment# run as follows: sh DrAdminRun <server port> <trace spec>export CLASSPATH=/usr/jdk_base/lib/classes.zip:/usr/WebSphere/AppServer/lib/ujc.jar:/usr/WebSphere/AppServer/lib/ejs.jar:/usr/WebSphere/AppServer/lib/admin.jarecho $CLASSPATHechoechojava -classpath $CLASSPATH com.ibm.ejs.sm.util.debug.DrAdmin -serverPort $1 -setTrace $2` **Note:** Verify the CLASSPATH is correct for your environment. The script example was written for AIX. You must change the CLASSPATH for Solaris.
2. Invoke DrAdminRun with the port number and the trace string. Use the port number from DrAdmin entry in the trace file, xxxx. Your input from a command prompt will be: `sh DrAdminRun xxxx com.ibm.ejs.*=all=enabled`
3. Start the administrative client with the debug option by invoking adminclient.sh from the WAS_ROOT bin directory: `adminclient.sh debug`

## 8.4.2: Enabling and reading ORB trace

In this section you will find information on how to read and enable ORB trace.

## Reading ORB trace

In order to read ORB trace, you need to understand the ORB communications log.

### ORB communications log

The ORB communications log, typically referred to as CommTrace, contains the sequence of GIOP messages sent and received by the ORB during application execution. It might be necessary to understand the low-level sequence of client-to-server or server-to-server interactions during problem determination. This section uses trace entries from a sample log to explain the contents of the log and help you understand the interaction sequence. It focuses only in the GIOP messages and does not discuss in detail additional trace information which appears when intervening with the GIOP-message boundaries.

The Sample Log Entry - GIOP Request and Sample Log Entry - GIOP Reply illustrate typical log entries. The entries have been annotated with line numbers for easy reference.

### Enabling CommTrace

The ORB property com.ibm.CORBA.CommTrace is used to enable/disable recording of trace entries during execution. Trace entries are recorded when the property is set true. In addition, the property com.ibm.CORBA.Debug must also be set true.

### Identifying start of a GIOP messages

The start of a GIOP message is identified by a line which contains either "OUT GOING:" or "IN COMING:" depending on whether the message is a request message or reply message.

Following the identifying line entry is a series of items, formatted for convenience, with information extracted from the raw message that identify the endpoints in this particular message interaction. See lines 3-12 in both figures. The formatted items include:

- GIOP message type, e.g. "Request Message", "Reply Message", in line 3
- Date and time message was recorded, in line 4
- Information useful in uniquely identifying the thread in execution when the message was recorded, along with other thread-specific information, in line 5
- The local and remote TCP/IP ports used for the interaction, in lines 6-9
- The GIOP version, byte order and message size, in lines 10-12

### Service context information

Following the introductory message information, the service contexts in the message are also formatted for convenience. Each GIOP message might contain a sequence of service contexts sent/received by each endpoint. Service contexts, identified uniquely with an ID, contain data used in the specific interaction, such as security, character codeset conversion and ORB version information. The content of some of the service contexts is standardized and specified by the OMG, while other service contexts are proprietary and specified by each vendor. IBM-specific service contexts are identified with ID's which begin with 0x4942.

Lines 14-33 in Sample Log Entry - GIOP Request and Sample Log Entry - GIOP Reply illustrate typical service context entries. There are three service contexts in both the request and reply messages, as shown in line 14. The ID, length of data, and raw data for each service context is printed next. Lines 15-17 show an IBM-proprietary context, as indicated by the ID 0x49424D12. Lines 18-33 show two standard service contexts, identified by ID 0x6 (line 18) and 0x1 (line 31). Refer to the CORBA specification for the definition of the standardized service contexts.

Service context 0x1 (CORBA::IOP::CodeSets) is used to publish the character codesets supported by the ORB in order to negotiate and determine the codeset used to transmit character data; service context 0x6 (CORBA::IOP::SendingContextRunTime) is used by RMI-IIOP to provide the receiving endpoint with the IOR for the SendingContextRuntime object; and IBM service context 0x49424D12 is used to publish ORB PartnerVersion information in order to support release-to-release inter-operability between sending and receiving ORBs.

### Request ID, response expected and reply status

The request ID is an integer generated by the ORB. It is used to identify and associate each request with its corresponding reply. This is necessary because the ORB can receive requests from multiple clients and must be able to associate each reply with the corresponding originating request.

Lines 34-35 in Sample Log Entry - GIOP Request show the request ID, followed by an indication to the receiving endpoint that a response is expected (CORBA allows sending of one-way requests for which a response is not expected.)

Lines 34-35 in Sample Log Entry - GIOP Reply show the request ID, followed by the reply status received after completing the corresponding previously sent request. Line 35 shows the status of "LOCATION_FORWARD", which indicates to the sending endpoint that the request needs to be re-issued and forwarded to a different object. The message body contains the IOR for the new object. The forwarding action is done automatically by the ORB and is transparent to the client sending the request.

### Object Key

Lines 36-42 in Sample Log Entry - GIOP Request show the object key, the internal representation used by the ORB during execution to identify and locate the target object intended to receive the request message. Object keys are not standardized.

### Operation

Line 43 in Sample Log Entry - GIOP Request shows the name of the operation to be executed by the target object in the receiving endpoint. In this sample the specific operation requested is named "retrieve."

### Principal identifier

Lines 44-46 in Sample Log Entry - GIOP Request show the length and contents of the CORBA object known as "CORBA::Principal" used by the CORBA Security Service to identify security credential information of the sender.

### Data offset

Line 47 in Sample Log Entry - GIOP Request and line 38 in Sample Log Entry - GIOP Reply show the offset, relative to the beginning of the GIOP message, where the remainder body of the request or reply message is located. This portion of the message is specific to each operation and varies from operation to operation. Therefore, it is not formatted, as the specific contents are not known by the ORB.

The offset is printed as an aid to quickly locating the operation-specific data in the raw GIOP message dump, which follows the data offset.

### Raw GIOP message dump

Starting at line 50 in Sample Log Entry - GIOP Request and line 41 in Sample Log Entry - GIOP Reply a raw dump of the entire GIOP message is printed in hexadecimal format. Request messages contain the parameters required by the given operation and reply messages contain the return values and content of output parameters as required by the given operation. Not all of the message raw data has been included in the figures for brevity.

## Sample ORB communications log entries

### Sample Log Entry - GIOP Request

```
OUT GOING:


Request Message
Date: April 18, 2001 10:14:21 AM EDT
Thread Info: P=259545:O=0:CT
Local Port: 65454 (0xFFAE)
Local IP: njros1un1801.prudential.com/48.113.114.2
Remote Port: 9000 (0x2328)
Remote IP: njros1un1801.prudential.com/48.113.114.2
GIOP Version: 1.1
Byte order: big endian
```

```
Message size: 380 (0x17C)
--
Service Context: length = 3 (0x3)
Context ID: 1229081874 (0x49424D12)
Context data: length = 8 (0x8)
00000000 000C0001
Context ID: 6 (0x6)
Context data: length = 168 (0xA8)
00000000 00000028 49444C3A 6F6D672E
6F72672F 53656E64 696E6743 6F6E7465
78742F43 6F646542 6173653A 312E3000
00000001 00000000 0000006C 00010100
0000000D 34382E31 31332E31 31342E32
0000FFAF 0000002C 4A4D4249 00000010
42F65A47 33623030 30303030 30303030
30303030 00000024 00000008 00000000
00000000 00000001 00000001 00000018
00000000 00010001 00000001 00010020
00010100 00000000
Context ID: 1 (0x1)
Context data: length = 12 (0xC)
00000000 00010001 00010100
Request ID: 5 (0x5)
Response is expected? Yes.
Object Key: length = 87 (0x57)
4A4D4249 00000012 33C5F0DD 31303030
30303030 30303030 30303030 00000024
00000033 49454A50 01000D5F 5F61646D
696E5365 72766572 0F747261 6E4C6F67
```

```
53696D70 6C654F41 0000000B 7472616E
4C6F6757 697265
Operation: retrieve
Principal: length = 32 (0x20)
49424D44 3A000000 0000000D 34382E31
31332E31 31342E32 00000000 00000000
Data Offset: 17c


0000: 47494F50 01010000 0000017C 00000003 GIOP.......|....
0010: 49424D12 00000008 00000000 000C0001 IBM............
0020: [remainder of message body deleted for brevity]
```

### Sample Log Entry - GIOP Reply

```
IN COMING:


Reply Message
Date: April 18, 2001 10:14:21 AM EDT
Thread Info:
P=259545:O=0:StandardRT=0:LocalPort=65454:RemoteHost=48.113.114.2:RemotePort=9000:
Local Port: 65454 (0xFFAE)
Local IP: njros1un1801.prudential.com/48.113.114.2
Remote Port: 9000 (0x2328)
Remote IP: njros1un1801.prudential.com/48.113.114.2
GIOP Version: 1.1
Byte order: big endian
Message size: 396 (0x18C)
--
Service Context: length = 3 (0x3)
Context ID: 1229081874 (0x49424D12)
Context data: length = 8 (0x8)
00000000 000C0001
Context ID: 6 (0x6)
Context data: length = 168 (0xA8)
00000000 00000028 49444C3A 6F6D672E
6F72672F 53656E64 696E6743 6F6E7465
78742F43 6F646542 6173653A 312E3000
00000001 00000000 0000006C 00010100
0000000D 34382E31 31332E31 31342E32
0000FFAF 0000002C 4A4D4249 00000010
42F65A47 33623030 30303030 30303030
30303030 00000024 00000008 00000000
00000000 00000001 00000001 00000018
00000000 00010001 00000001 00010020
00010100 00000000
Context ID: 1 (0x1)
Context data: length = 12 (0xC)
00000000 00010001 00010100
Request ID: 5 (0x5)
Reply Status: LOCATION_FORWARD
Object Key: length = 1 (0x1)
00
Data Offset: f1


0000: 47494F50 01010001 0000018C 00000003 GIOP............
0010: 49424D12 00000008 00000000 000C0001 IBM............
0020: [remainder of message body deleted for brevity]
```

# Enabling ORB trace

Below, you will find instructions for enabling ORB trace in the WebSphere Administrative Server, WebSphere Application Server, administrative client (console) on Windows NT, and the administrative client (console) on Unix.

## Tracing the WebSphere Administrative Server

Follow these steps:

1. Make sure the default server and administrative server are not running.
2. Make a backup copy of the admin.config file.
3. Add the following lines to the admin.config file:
    - com.ibm.CORBA.Debug=true
    - com.ibm.CORBA.CommTrace=true
    - com.ibm.ejs.sm.adminServer.traceString="ORBRas=all=enabled"
    - com.ibm.ejs.sm.adminServer.traceOutput=c\:/tracedirectory/adminserver.trace**NOTE:** On Unix the directory path would look more like /opt/tracedirectory or /usr/tracedirectory)
4. Start the administrative server.
5. The resulting trace file is ==> c\:/tracedirectory/adminserver.trace.

## Tracing the WebSphere Application Server (default server)

Follow these instructions:

There is a checkbox on the ORB configuration property sheet which is accessible from the Services tab of the application server property sheet in the administrative console. When that checkbox is enabled, ORB communication trace is configured for that application server.

If there is already a traceOutput file defined for this application server, then the communicationtrace output is directed to that file. If there is no output file defined, the file"$WAS_HOME/logs/<server name>.trace" is defined to contain the communication trace output.

## Tracing the administrative client (console) on Windows NT

Follow these instructions:

1. Go to the WebSphere/AppServer/bin subdirectory and make a backup copy of adminclient.bat file.
2. Edit the adminclient.bat file for the following:
   Change

```
goto NODEBUG
:DEBUG
set DEBUGOPTS=-traceString "com.ibm.*=all=enabled"
```

to

```
goto NODEBUG
:DEBUG
set DEBUGOPTS=-traceString
"com.ibm.*=all=enabled:ORBRas=all=enabled"
```

3. Add the two trace parameters to the following "%JAVA_HOME%\bin\java" statement:
   - -Dcom.ibm.CORBA.Debug=true
   - -Dcom.ibm.CORBA.CommTrace=true

   The statement should be in one continuous line. Add "%DEBUGOPTS%" also to the statement if it does not already exist.

   If "%DEBUGOPTS%"=="" does exist, go to START

   ```
   %JAVA_HOME%\bin\java -Dcom.ibm.CORBA.Debug=true
   -Dcom.ibm.CORBA.CommTrace=true -Xminf0.15 -Xmaxf0.25 -classpath %WAS_CP%
   %CLIENTSAS%
   -Dcom.ibm.CORBA.principalName=%COMPUTERNAME%/AdminClient
   -Dserver.root=%WAS_HOME% com.ibm.ejs.sm.client.ui.EJSConsole %DEST%
   %DESTPORT% %DEBUGOPTS% %QUALIFYNAMES%
   ```

   Go to END
4. After the administrative server has been started, using the statement "adminclient debug > adminclientttrace" from WebSphere/AppServer/bin subdirectory.
5. The resulting trace file is adminclienttrace.

### Tracing administrative client(console) from Unix

Follow these instructions:

1. Go to WebSphere/AppServer/bin subdirectory and make a backup copy of adminclient.sh.
2. Edit the adminclient.sh for the following:

   Change

   ```
   elif [ "$1" = "debug" ]
   ```

   ```
   thenDEBUGOPTS='-traceString "com.ibm.*=all=enabled" '
   ```

   to

   ```
   elif [ "$1" = "debug" ]
   ```

   ```
   thenDEBUGOPTS='-traceString
   ```

```
"com.ibm.*=all=enabled:ORBRas=all=enabled" '
```

Add the three trace parameters to the "$JAVA_HOME/bin/java" statement. If "$DEBUGOPTS" is already in the statment, then there is no need to add it again.
-Dcom.ibm.CORBA.Debug=true -Dcom.ibm.CORBA.CommTrace=true $DEBUGOPTS

3. After the administrative server has been started, using the statement "adminclient.sh debug 2>&1 | tee adminclienttrace" from WebSphere/AppServer/bin subdirectory.
4. The resulting trace file is adminclienttrace (in the bin directory).

## 8.4.2: ORB request trace

ORB request trace can be enabled on the server to display the target object type/class and the function name. It is a quick way to let you view function call flow and understand how objects work with each other. ORB communication tracing is most appropriate when you want detailed information such as input and output parameters.

This section describes how to turn on the ORB request trace and interpret thetrace output:

- Setting the ORB request trace
- Sample: Formatted ORB request trace output

## Setting the ORB request trace

To set the ORB request trace, perform these steps:

1. Display the system manager user interface, and set the view level to Control.
2. Expand your Host Images.
3. Expand the Server Images folder.
4. Left click on your server image to see if it is running. The status bar at thebottom of the System Manager user interface application displays the state (and health) of the selected server. Note: You do not have to stop theserver to set this trace.
5. Right-click on your server image and select Properties. This displays theProperties Editor for the Server Image.
6. In the Properties Editor window, click the Component Trace tab.
7. Set the ORB request trace level attribute value to Advanced. Click **Apply**and then **OK** to enable the trace.

This enables trace information to be collected over a period of time into filesin the subdirectory service\serveServerName. Refer to showlog utility for more information on formatting trace logs. You can use the Log Analyzer to view the output of showlog.

## Sample: Formatted ORB request trace output

The formatted output file looks like the formatted activity log. Function callsare recorded in the activity log. The function name can be seen in thefunctionName or PrimaryMessage fields. Here is an example of the contentsof a formatted output file:

```
ComponentId: 393319

ProcessId: 567

ThreadId: 534

FunctionName:
CORBA::BOA::local_object_to_object_key(CORBA::Object_ORBProxy_ptr)

ProbeId: 2990

SourceId: 1.66 src/orb/src/somd/boa.cpp

Manufacturer: IBM
```

```
Product: Component Broker
Version: 1.3
SOMProcessType: 5
ServerName: PersonServer
clientHostName:
clientUserId:
TimeStamp: 10/6/98 9:54:14.343609431
UnitOfWork:
Severity: 3
Category: 3
FormatWarning: 0
PrimaryMessage: The function
CORBA::BOA::local_object_to_object_key(CORBA::Object_ORBProxy_ptr):2990
reported data.
ExtendedMessage:
RawDataLen: 0
----------------------------------------------------
ComponentId: 393319
ProcessId: 567
ThreadId: 534
FunctionName: CORBA::Request::send_deferred()
ProbeId: 1405
SourceId: 1.57.1.2 src/orb/src/request/request.cpp
Manufacturer: IBM
Product: Component Broker
Version: 1.3
SOMProcessType: 5
ServerName: PersonServer
clientHostName:
clientUserId:
TimeStamp: 10/6/98 9:54:14.371803789
```

```
UnitOfWork:

Severity: 3

Category: 3

FormatWarning: 0

PrimaryMessage: The function CORBA::Request::send_deferred():1405
reported data.

ExtendedMessage:

RawDataLen: 0
```

---

```
ComponentId: 393319

ProcessId: 567

ThreadId: 534

FunctionName: CORBA::Request::invoke()

ProbeId: 1338

SourceId: 1.57.1.2 src/orb/src/request/request.cpp

Manufacturer: IBM

Product: Component Broker

Version: 1.3

SOMProcessType: 5

ServerName: PersonServer

clientHostName:

clientUserId:

TimeStamp: 10/6/98 9:54:16.488164076

UnitOfWork:

Severity: 3

Category: 3

FormatWarning: 0

PrimaryMessage: The function CORBA::Request::invoke():1338 reported
data.

ExtendedMessage:

RawDataLen: 0
```

## 8.5: Identifying the problem

### Available tools, traces and logs for specific problems

**Problem/Tool/Trace/Log Matrix**

| Problem type | Tool | Trace/Log | Description | Location |
|---|---|---|---|---|
| **Install failure** | | NT wssetup.log<br>UNIX WebSphere.instl | Traces install events and settings | NT <WebSphere/AppServer/logs> UNIX See article Viewing logs fordirectory information |
| **Startup failure** | jdbctest.java | | Tests jdk settings and connectivity | Invoke jdbctest.java tool from command prompt |
| **Administrative server startup/shutdown failures** | Events Viewer,<br>ShowCfg servlet | trace file and<br>UNIX nanny trace | Displays fatal errors during startup or shutdown<br>Displays configuration information | <WebSphere/AppServer/logs> |
| **Application Server startup failure** | Events Viewer,<br>ShowCfg servlet | trace file | Displays fatal errors during startup<br>Displays configuration information | <WebSphere/AppServer/logs> |
| **Non-startup server problems** | Events Viewer | trace file | Traces runtime problems | <WebSphere/AppServer/logs> |
| **Administrative server not responding** | DrAdmin | | Tracing service used primarily to dump thread stacks | <WebSphere/AppServer/bin/debug> |
| **Runtime** | Log Analyzer | logs/activity.log | Displays and analyzes runtime errors | Invoke from <*product_installation_root*>/bin/waslogbr.bat\|sh |
| **Database problems** | jdbctest.java | NT wasdb2.log | jdbctest.java tests database connectivity<br>wasdb2.loglists database configuration problems | Invoke jdbctest.java tool from command prompt<br><WebSphere/AppServer/logs> |
| **Servlet/EJB/JSP problems** | Object Level Trace | | Object Level Tracing and Debugging | Enable OLT through the OLT Controller |

| | | | | |
|---|---|---|---|---|
| **Servlet/EJB/JSP problems** | Distributed Debugger,<br>HitCount servlet,<br>Snoop servlet | stdout and stderr | Identify application problems using Debugger<br>Application server stdout and stderr logs, and servlets | Review the Debugger documentation for implementation instructions.<br>Trace directory is <WebSphere/AppServer/logs>. |
| **Communication problems** | Java<sup>TM</sup> Socket Level Trace | | Describes ORB communication problems over heterogeneous networks via IIOP | Invoke socktrace tool from command prompt |
| **Name space problems** | Java<sup>TM</sup> Name Tree Browser | | Displays elements in WebSphere Application Server name space | Invoke jntb tool from command prompt |
| **Performance problems** | NT WebSphere Resource Analyzer | | Describes how to monitor and tune WebSphere Application Server | Review the documentation for implementation instructions. |

## 8.5.1: Plug-in problems

HTTP servers are legacy Web products, created at a time when theywere the only conduit between browsers and HTML or CGI files. With the evolution of Web technology, users now require servers to handle servlets, JSP files and EJBs.WebSphere Application Server supports this technology, but to provide these functions it mustintercept requests sent to the HTTP Server.

The plug-in component extends the function of the HTTP Server by interceptingrequests and passing them to either WebSphere Application Server or the HTTP Server. The following three files in the `<WAS_root>temp` directory allow the plug-in to determine the request's destination:

- `<WAS_root>/temp/`**`rules.props`**
    - Provides a snapshot of the existing topology and lists available Web resources and pathsto handle service requests
- `<WAS_root>/temp/`**`vhosts.props`**
    - Provides virtual hosting information that is transferred to the WebSphere ApplicationServer runtime environment
- `<WAS_root>/temp/`**`queues.props`**
    - Provides names of links to different servlet engines. The number of links listed inthis file vary according to the number of application servers, clones and other servlet engine resources that are defined in the product.

This is the high level view of the plug-in process flow:

```
 Browser -->  WebSphere plugin -->  HTTP Server or WebSphere Application Server
```

### Typical plug-in problems

Generally, plug-in failures are caused by missing files or an incorrectly configured HTTP Server.

To diagnose plug-in problems, verify the data in the files are consistent with both the HTTP request and the active configuration in the servlet engine. The plug-in configuration files are generated periodically so a delay can occur between the time a change is made in the system andthe time the change is reflected in the configuration files.

The following error descriptions are symptoms of plug-in problems:

1. Servlet requests are not fulfilled. Verify the following to determinethe cause of the problem:
    - The Web server can serve HTML pages
    - The administrative console can connect to the Web server
    - The Default server is started
    - Ensure the Web server hostname and port number are identical to the ones definedin the virtual host's alias table.
    - Ensure the appropriate `.DLL` file for the Web server is present in the `<WebSphere-root>bin`
2. Pipe broken messages appear. Verify the following to determine the cause ofthe problem:
    - TCP/IP connection exists between Web server and WebSphere Application Server.
    - No process or thread failures occurred.
    - No access violations occurred.

### How to debug plug-in problems

Check for errors in the following logs, and in the trace file:

- trace.log.<http server>.<date>
- <AppServer>_stderr.log
- <AppServer>_stdout.log
- <AppServer>_native.log.<date>

See files 8.3: Logs and 8.4: Trace for more information on logs and trace file.

If there are no entries in the logs or trace file, comment out the WebSphere ApplicationServer plugin in the httpd.conf file.This will help you determine if the failure originates with WebSphere Application Server or the HTTP Server.

The WebSphere plugin property in the httpd.conf file is:

```
Load Module ibm_app_server_module
```

Restart the HTTP Server. If the Web Server initializes and runs, then WebSphere Application Server has a problem.

## 8.5.2: Servlet redirector problems

Servlet redirectors are used to separate the HTTP server from the WebSphere Application Server.There are different types of redirectors:

- Thick - servlet redirector resides on the same machine as WebSphere Administrative Server.
- Thin - servlet redirector runs on a separate machine from the WebSphere Administrative Server.
  A *thin* servlet redirector is useful in network configurations where the HTTP server is outside a firewall but WebSphere Application Server isbehind a firewall, or where WebSphere Application Server is located in the DMZ, a machine located between twofirewalls.

  > **Note:** If the Web server, application server and the server handling servlet requestsall reside on the same machine, use Open Servlet Engine(OSE), instead of a servlet redirector

See file, Entry point to servlet redirector configuration, for information on configuringservlet redirectors.

### Key features

The key features of servlet redirectors are:

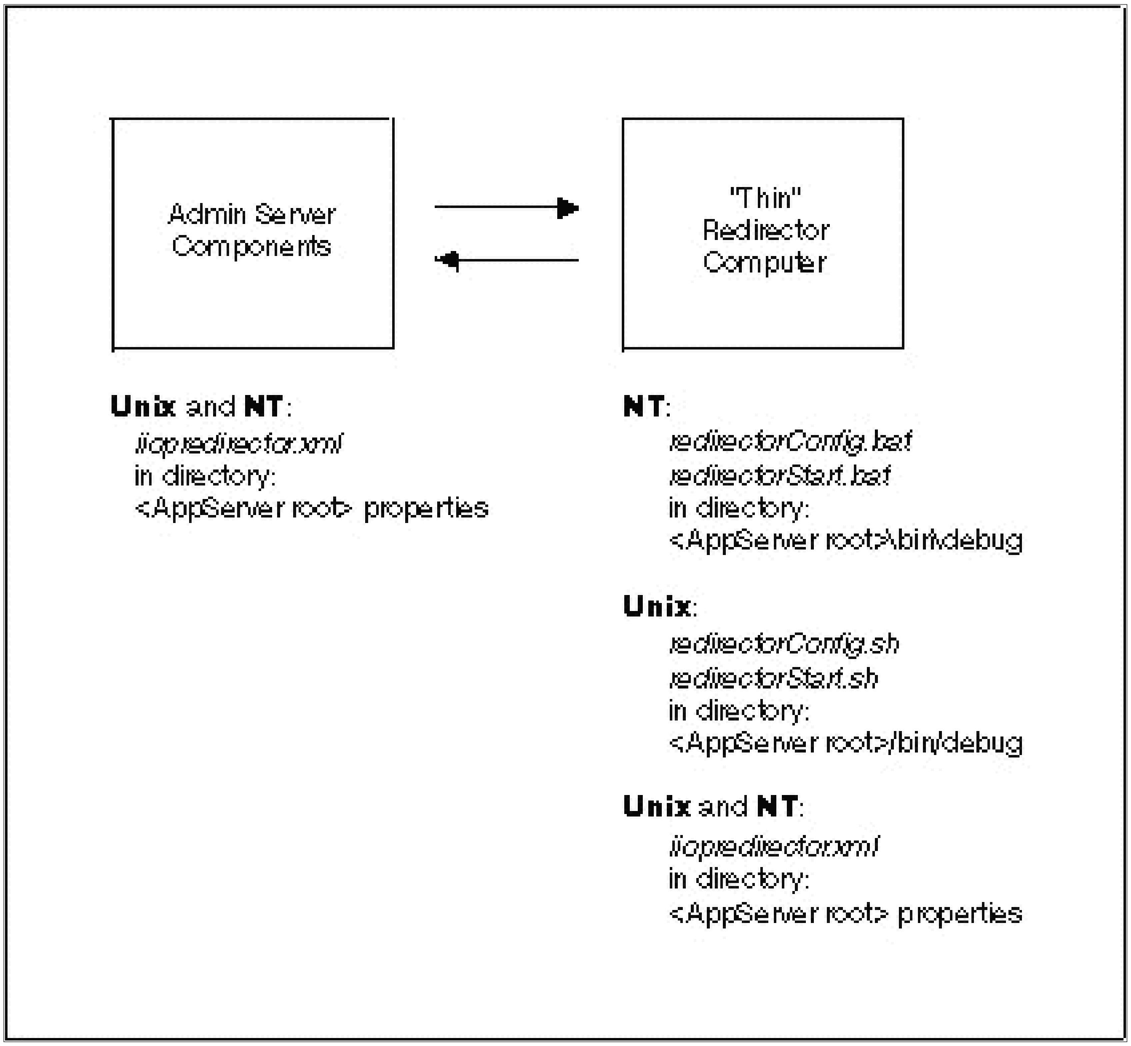
1. Use Internet Inter-Orb Protocol (IIOP) for communication
2. Are initialized by copying the queues.properties, rules.properties,and vhosts.properties files from the WebSphere Application Server machine to the servlet redirector machine.
3. Use the following ports to transfer the properties files:
   - port 900 - bootstrap port
   - port 9000 - name services port
   - redirector listener port
4. Require the receiving RemoteSRP bean to be running on the WebSphere Application Server
5. When servlet redirectors are running, beside the ports previously listed, they alsorequire the following, additional ports:
   - Application Server listener port
   - Admin server nanny listener port
   - *Thick* servlet redirectors also require repository database connection ports

## Typical problems

Typical problems with servlet redirectors are:

1. **Error 404** - URL not found appears on browser when accessing a servlet.
   The trace log for the HHTP server contains entry:
   *Failure while locating a vhost for <server>*
   This error may occur if:
   - The short (myserver) and fully qualified (myserver.mydomain.com) host names of the HTTP server are not in the virtual host alias list.
   - A port other than the standard port 80 is used and that port is not in the virtual host alias list.
2. CORBA.COMM_FAILURE appears when running the thin servlet redirector. Thiserror may occur if:
   - Syntax error exists in batch file
   - Host name/port number is not in DNS or hosts.properties file on servlet redirector machine
   - Hostname portis not in DNS or hosts.properties file on WebSphere ApplicationServer machine
   - -ListenerPort parameter value is not unique
   - Syntax error in -ListenerPort parameter
3. Unable to initialize threads: (null) error when running the batch/script fileto configure the plug-in files.This error may occur if:
   - Some JDK other than the IBM supported JDK is in the classpath ahead of the supported JDK. To resolvethis problem, install the supported IBM JDK, and change the path and classpath to point to it first.
   - Path does not point to java.exe.
4. When starting servlet redirector with batch/script file, get error message "Error locating Remote SRP Home - Attribute Not Set." This error occurs because:
   - RemoteSRP bean was not added and started on WebSphere Application Server
5. Do not use servlet redirectors if you must:
   - Enable security on your thin servlet redirector machine
   - Provide the real Web browser on your thin servlet redirector machine
   - Use the Network Address Translation feature of firewalls so that the internal address of WebSphere Application Server is not available outside the firewall

   **Reminder:** Generally errors occur configuring the thin servlet redirector function because required configuration files are missing. A thinservlet redirector machine requires the following shell script files (on UNIX platforms), or bat fileson Windows NT as well as the iiopredirector.xml file to enable the servlet redirector function.



# 8.5.3: Workload management/cloning/remote administration problems

Unlike WebSphere Application Server Advanced Edition, the Standard Edition does not support the workload management function. WebSphere Application Server Standard Edition is limited to a single physical server. However, Standard Edition provides multiple JVMs that can be mappedto multiple virtual hosts on a single HTTP Server. Therefore multiple Web sites can still be hosted using one Standard Edition Application Server.

## Remote administration

You can remotely administer WebSphere Application Server using:

- Remote administrative console
- X Windows clients on UNIX machines
- Web based WebSphere administrative console

### Remote administrative console

Use can run admin console remotely using the adminclient.bat file on Windows NT, oradminclient.sh file on UNIX. See article Administrative models for more information on implementing a remoteadministrative client.

Typical remote admin console problems are:

- The com.ibm.ejs.util.cache.FaultException error occurs in the stack trace becausethe JDK running on the client machine cannot communicate with the JDK running on the administrative servermachine. The resolution is to upgrade the backlevel JDK.
- The Network Address Translation (NAT) function in firewalls cannot be used with a remote administrative client.The internal address of the administrative server is not recognized by the administrative client outside thefirewall. No circumvention exists for this problem.

### X Windows clients on UNIX machines

X windows client software can run on any platform but requires a UNIX X Windows server.(Currently the X Windows server is only available on AIX and Solaris platforms.)

Typical X Windows client problems are:

- You cannot run an administrative console remotely through the X Windows client using anauthorized, non-root account with global security enabled. The error message, FATAL Could not bind to the Administrative Server on {0}{1}, appears onthe screen when the adminclient.sh or .bat file is executed. No circumvention exists for this problem.

### Web-based administrative console

See article Web administrative console overview for informationon configuring and implementing a Web-based administrative console.

## 8.5.4: Installation problems

Succesful installation means that no errors occur during the install process and,more importantly, that the product runs correctly the first time you start it.

Installation and start-up problems occur for one of the following reasons:

- Database is not configured properly
- Classpath is incorrect
- Administrative server fails to start

### Install options

WebSphere Application Server provides a Java TM Graphical User Interface (GUI)install that is available on all platforms, and a native install that is available on the UNIX platforms (AIX, Solaris, HP).

> **Note:** If you used the native install option to install WebSphere Application Server on a UNIX platform, you **must** also ***uninstall***using the native uninstall option. In other words, you cannot do a nativeinstall and a Java GUI uninstall.

Follow the steps in one of the "platform specific" install guides to install the product. These install guides are available from the InfoCenter, in the Selecting installation steps section.

### Database configuration problems

If the database is not configured properly, installationof WebSphere Application Server will fail. If specific WebSphere components did install but the database is misconfigured, the product will not run properly.

Starting WebSphere Application Server with an improperly configured database will generate the following error messages and exceptions:

```
   Establishing connection please wait ...  Error - could not get attributes
com.ibm.ejs.util.cache.FaultException at java.lang.Throwable<init>
com.ibm.ejs.sm.client.ClientException getAttributeFailure Attributes may be involved
com.ibm.ejs.sm.client.RpositoryOpException could not get attributes
```

### Classpath problems

The classpath provides the JavaTM runtime environment for the following WebSphere Application Server processes:

- Administrative service - the backend process for system management
- Administrative console - the Graphical User Interface (GUI) used for system management
- One or more application servers - each application server consists of multiple containers for deployment of Enterprise Java Beans (EJBs) and one servlet engine for deployment of Web applications
- Nanny service (on UNIX platforms only) - a daemon that monitorsthe administrative service. The nanny service starts the administrative serviceinitially and restarts it if it fails.

Each of these processes runs in its own Java Virtual Machine (JVM). The classpath for each process tells that process where to search for classes. The classpath can be set:

- In an administrative service startup script
- In an admin.config file
- With application server command line arguments
- By Web applications

#### Classpath properties

Each process has an associated set of properties ("Java-speak" for environment variables).These properties are defined in the admin.config file that is located in directory:

`<WebSphere root> bin`

The applicable properties in admin.config are:

- com.ibm.ejs.sm.util.process.Nanny.maxtries
- com.ibm.ejs.sm.adminserver.classpath
- com.ibm.ejs.sm.util.process.Nanny.path
- com.ibm.ws.jdk.path

The classpath settings in the admin.config file apply to the administrativeservice, and they are also inherited by all other WebSphere Application Server processes.

For more information on these and other WebSphere Application Server properties, see file,6: Administer applications.

### Classpath failures

Typical classpath failures are:

1. When a servlet class is missing from a Web application classpath, the following errors occur:
    - In a browser window, the browser displays error ***Error 500*** with message,"Failed to load target servlet [snoop]."
    - Browser stack trace and <AppServerName>_stderr.log show java.lang.ClassNotFoundException
    - <AppServerName>_stdout.log shows javax.servlet.ServletException
2. When utility classes, such as dates.class or time.class, are in a Web application's classpath, the following errors occur:
    - Browser shows error messagejava.lang.VerifyError
    - Verbose JVM output written to the <AppServerName>_stderr.log shows java.lang.VerifyError: com/bcs/jsftest/test

      **Note:** A WebSphere Application Server problem exists onthe Windows NT platform which prevents the stderr buffer from being flushed until the applicationserver is stopped. No circumvention for this problem is available at this time.
3. When classes use Java Native Interface (JNI), the following errors occur:
    - <AppServerName>_stdout.log shows java.lang.UnsatisfiedLinkError

   To resolve the problem, do the following:

    - Ensure the shared libraries are available in the path statement on Windows NT. On UNIX, make sure the LD_LIBRARY_PATH is defined in file startupServer.sh.
    - Ensure that the property defined in com.ibm.ejs.sm.adminserver.classpath, in the admin.configfile, includes classes that make JNI calls into shared libraries.

## Administrative server problems

Successfully starting the administrative server not only indicates a successful install of WebSphere ApplicationServer, but it also means the following tasks were completed:

- System management repository tables were created in the database.
- Nodes and host aliases were created in the repository tables with xml.

- Default repository tables were created with xml.

Therefore, when the administrative server fails to start, it also means the installation of WebSphere Application Server is incomplete.

### Administrative server start failures

The administrative server fails to start for the following reasons:

1. The port is in use. See the port problems section for more information.
2. Another administrative server is running. The administrative server service in the Windows NT control panel or the startupServer.sh script on UNIX, is the same service/process as the one started through WebSphere\Appserver\bin\debug\adminserver.bat file on Windows NT or adminserver.sh on UNIX.
3. The WebSphere Application Server database repository, (WAS on DB2 or ORCL on Oracle), is not created. The first time you start the administrative server process, it attempts to create the default configuration in the WAS or ORCL database. You will see a 2140 error message if the database is not created.
4. Connection to DB2 or Oracle fails. This also shows up as a 2140 error message. Ensure DB2 is running. Verify the connection to the WAS database is successful.
   To test the DB2 connection, from a DB2 command window, type:

   ```
   DB2 connect to was
   ```

   If you cannot connect to DB2, verify the following:
   - Ensure the right level of code is installed on the WebSphere Application Server machine
   - For a remote repository, ensure the DB2 client is configured properly to point to DB2 server for the WAS database.

   Perform the same tests for Oracle.
5. User ID does not have proper authority or access:
   - To ensure proper authority, follow the database configuration steps in the install guides.
   - In the UNIX environment, log on as ***root*** to start AdminServer.
   - In the Windows NT environment, verify the following conditions are true:
     - User is logged in as an administrative user
     - User name in security panel is correct
     - User is part of the administrator's group.
     - The Administrative server is registered as a service to NT. To manually add the administrative server as a service, from a command prompt, enter: <WebSphere\AppServer>\bin\adminservice.exe install <WebSphere\AppServer>\bin\admin.config <HostName>\<User> <Password>
     - User ID has proper rights to start the administrative server. If using a domain ID, start the administrative server with a local ID to see if the domain is the problem. To check a user's rights:
       1. From ***Start*** > Programs > Administrative Tools > User Manager
       2. Select ***Policies*** > ***User Rights***
       3. Check ***Show Advanced User Rights*** checkbox in lower left corner
       4. Add the following rights to the user ID:
          - Log on as a service
          - Act as part of the operating system

- If you change the Windows NT user ID/password but WebSphere Application Server is not updated, then the administrative server startup will fail.
  Update the user ID/password in the following areas:
  - In Windows NT services for the IBM WebSphere Administrative Server service:
    1. From ***Start*** > Settings > Control Panel, double click ***Services***
    2. Select ***IBM WebSphere Administrative Server***
    3. Click ***Startup***
    4. Change the user ID/password under this account
  - in admin.config (if the DB2 userid/password also changed)

### Port problems

WebSphere Application Server will fail to start if certain ports are in use. Typical port problem descriptions follow:

1. When the bootstrap port is in use, you may see the following error when starting WebSphere:

   ```
   009.765.6005c5b F Nameserver Failed to start the Bootstrap server
   org.omg.CORBA.INTERNAL: minor code: 8 completed: No
   ```

   This error is similar to the *Port 9000 in use* error when starting WebSphere Application Server.

   To fix the problem, change the bootstrap port (the default is 900) in file, ***admin.config***, using property name:

   **`com.ibm.ejs.sm.adminServer.bootstrapPort`**

   If this property does not exist in file ***admin.config***, add it.

2. Port 9000 is the default port of the Admin Server location service daemon. Port 9000 is also used by many system resources including AIX X-windows manager. If you see error message,

   ```
   Port 9000 in use-select another port
   ```

   when executing the ./startupServer.sh command on AIX, the administrative server process cannot start because port 9000 is in use. You can change the port the location service daemon listens on by:

   - specifying ***-lsdPort*** option on the admin server command line
   - setting ***com.ibm.ejs.sm.adminServer.lsdPort*** property in the *admin.config* file located in directory **<WAS_ROOT>\bin** on Windows NT and **<WAS_ROOT>/bin** on UNIX.

# 8.6: Diagnosing configuration and installation problems

WebSphere Application Server uses a database to store and share configuration information across nodes. Problems configuring the database are described in the installation problemssection.

Generally, if the database is not configured properly, the WebSphere Application Server installation process will fail. Configuration problems occur after the product is installed.

The following table describes common configuration problems. Select an entry for more information.

| Problem description | Cause of problem |
|---|---|
| Cannot retrieve data for a specific session | Incorrect use of the database as a session store:<br>● Datasource incorrectly created or configured |
| Servlet requests are not satisfied | Web server problems:<br>● Plug-in failure<br>● Virtual host configuration incorrect |
| Error 404 - URL not found occurs when accessing a servlet | HTTP Server hostname or port problem |
| Error 500 - Failed to load target servlet | Servlet missing from Web application classpath |
| FATAL - Could not bind to the administrative server error | Remote administration failure |
| Nanny process fails to start administrative server | Verify all installation steps were successful |

| | |
|---|---|
| Administrative server fails to start | Generally an installation setup problem |

# 8.7: Using application level facilities

WebSphere Application Server Standard Editiononly supports Web applications, not enterprise beans. WebSphere Application ServerAdvanced Edition supports both Web applications and enterprise beans.

For more information on enterprise beans and Web applications, see the file on developing applications.

Tools that are specifically designed to debug application, servlet and EJB problems include OLTand Distributed Debugger.OLT provides an object level trace. The Distributed Debugger allows you to set trace points in your code.

See the Problem/Tool/Trace/Log Matrix for more information on appropriatetools and traces.

Typical application and EJB problems are:

- Invoking a servlet from a browser window
- A modified servlet is not reloaded
- Incorrectly using of databases as a session store

## Invoking a servlet by its URL

The following example describes what you should enter in a browser window to invoke a servlet. Errors occur when you fail to include the webappdirectory in the path.

http://server_machine/webapp/examples/showCfg

The components of the URL are:

| server_machine | webapp | examples | showCfg |
|---|---|---|---|

| | | | |
|---|---|---|---|
| Name of the application server computer | Virtual directory of the Web application loader.<br>Do not create a webapp directory. This directory is defined for you by WebSphere Application Server. For more information on *webapp*, see the file on the programming model and environment. | Application Web path<br>This is a default WebSphere servlet Web path.You can create a directory by any name as long as it is defined in the Web application's category. | Servlet URL, not the name of the code.<br>In this example, the actual Servlet class name isServletEngineConfigDumper. |

The URL illustrated above is the URL for showCfg, one of the default servlets shipped with WebSphereApplication Server.

## Reloading servlets

In earlier versions of WebSphere Application Server, specific reload directories in the reload process had to be defined. Currently, the only reload requirement is to store servlet classes in the Web application category. That is, ensure all your servlets are handled in the context of the Web application loader. After you update your servlets, the Web application loader will automatically reload them for you.

If your servlet classes are installed in the context of the Web application loader, but are not being reloaded, ensure the Auto Reload property is set to true.Follow these steps to check the setting of the Auto Reload property:

1. From the WebSphere Administration Console, select your application (or default_app if you stored your servlets in the default directory structure).
2. From the Web Application:default_app panel, select the **Advanced** tab.
3. Verify that the Auto Reload is set to true.

## Incorrect use of a database as a session store

WebSphere Application Server makes JDBC calls, using a predefined JDBC driver, to communicatewith a database. Both the JDBC driver and datasources must be configured using the administrative console.

The following errors occur if a datasource is misconfigured or does not exist:

- The browser window displays Error 500 with the message: `java.lang.NullPointerException`
- The <App_Server>.stderr.log displays the message: `javax.naming.NameNotFoundException: jdbc/xxx`
- The <App_Server>.stdout.log displays the message: `Failure while creating connection COM.ibm.db2.jdbc.app.DB2Exception: [IBM] [CLI Driver] SQL1013NThe database alias name or database name "SAMPLE" could not be found. SQLSTATE=42705`

Database connectivity problems cause persistence exceptions. AnEJSPersistenceException error may indicate JDBC or connection problems:

1. An invalid JDBC driver will prevent access to jar and class files
2. Review the SQLSTATE:`COM.ibm.db2.jdbc.app.DB2Exception: [IBM][CLI Driver]SQL1224N A database agent could not be started...SQLSTATE=55032...`The SQLSTATE code of 55032 indicates the system is out of connections.

    **Note:** Not using connection pooling causes most problems for BMP type EJBs. Common symptoms include:

    - Performance problems connecting to the database
    - Running out of connections

    To resolve the problem:

    1. Increase the number of connections permitted by DB2 or Oracle.
    2. On AIX, catalog the database as if it were remote.
    3. Ensure you close connections when programming exceptions occur.
    4. Verify that connections obtained in one method, are returned to the pool via close().
    5. Verify that your application does not try to access pre-empted connections (idle connectionsthat are now used by other

resources).

3. A database init failure could indicate the database does not exist:`com.ibm.ejs.persistence.EJSPersistenceException: Database init failure:`Nested exception is:`COM.ibm.db2.jdbc.app.DB2Exception: [IBM][CLI Driver]SQL1013N The database alias name or database name "YYY" could not be found...SQLSTATE=42705...`The SQLSTATE code of 42705 indicates the database does not exist or the server cannotconnect to it.

- 8.5: Identifying the problem

## 8.7.1: ORB-related minor codes

This document provides explanations of the minor error codes used by the WebSphere Application Server Advanced Edition Java ORB. These minor codes are not CORBA-compliant. CORBA-compliant minor codes usually begin with an OMG-assigned identification code, which consists of the vendor ID and digits that identify the minor code. However, the Java ORB minor codes do not contain the vendor ID.

Minor codes are associated with CORBA exceptions and provide greater detail about the errors that can occur. There is not a one-to-one mapping of exception names to minor codes. Instead, a minor code can be associated with several exception names. A minor code message can have different meanings depending on the exception that was thrown.

Minor codes are scoped to system exceptions in the range 0 to 4095. A minor code ID must be a unique number within the scope for each system exception, but there is no restriction that minor codes be unique across all system exceptions.

The following table lists the system exceptions and the corresponding minor error codes, where:

- Minor code: The minor error code
- Static variable: The name of the static variable for the minor error code
- Explanation: A description of the problem that caused the error
- User response: Actions to resolve the problem

**org.omg.CORBA.BAD_PARAM**

- Minor code: 1
- Static variable: com.sun.rmi.util.MinorCodes.NULL_PARAM
- Explanation: A parameter with a value of NULL was received. The parameter is not valid.
- User response: Ensure that parameters are initialized correctly.

**org.omg.CORBA.COMM_FAILURE**

- Minor code: 1
- Static variable: com.sun.rmi.util.MinorCodes.CONNECT_FAILURE
- Explanation: The ORB could not establish a connection to the server on the host and port that was identified by the object reference.
- User response: Ensure that the server is running and listening on the designated host and port.
- Minor code: 2
- Static variable: com.sun.rmi.util.MinorCodes.CONN_CLOSE_REBIND
- Explanation: A client request could not be processed, because the server had notified the client to close the connection and a new connection could not be established.
- User response: Ensure that the server is running and try the request again.

- Minor code: 3
- Static variable: com.sun.rmi.util.MinorCodes.WRITE_ERROR_SEND
- Explanation: An error was encountered while writing the request to the output stream.
- Minor code: 4
- Static variable: com.sun.rmi.util.MinorCodes.GET_PROPERTIES_ERROR
- Explanation: An exception was encountered while reading the initial services from a URL.
- User response: Ensure that the initial services URL is valid.
- Minor code: 6
- Static variable: com.sun.rmi.util.MinorCodes.INVOKE_ERROR
- Explanation: The ORB was unable to successfully connect to the server after several attempts.
- User response: Ensure that the server is running.

**org.omg.CORBA.DATA_CONVERSION**

- Minor code: 1
- Static variable: com.sun.rmi.util.MinorCodes.BAD_HEX_DIGIT
- Explanation: The object reference in string format contains at least one hexadecimal character that is not valid.
- User response: Obtain the original object reference and reformat it as a string using the object_to_string method on the ORB.
- Minor code: 2
- Static variable: com.sun.rmi.util.MinorCodes.BAD_STRINGIFIED_IOR_LEN
- Explanation: The length of the string-formatted object reference is not valid.
- User response: Obtain the original object reference and reformat it as a string using the object_to_string method on the ORB.
- Minor code: 3
- Static variable: com.sun.rmi.util.MinorCodes.BAD_STRINGIFIED_IOR
- Explanation: The string-formatted object reference is not valid.
- User response: Obtain the original object reference and reformat it as a string using the object_to_string method on the ORB.
- Minor code: 4
- Static variable: com.sun.rmi.util.MinorCodes.BAD_MODIFIER
- Explanation: The initial reference could not be resolved, because the host or the port is not valid or was not specified.
- User response: Specify the correct host and port.
- Minor code: 5
- Static variable: com.sun.rmi.util.MinorCodes.CODESET_INCOMPATIBLE

- Explanation: While processing the service context code sets for a request, an incompatible code set was encountered.

**org.omg.CORBA.INTERNAL**

- Minor code: 8
- Static variable: com.sun.rmi.util.MinorCodes.CREATE_LISTENER_FAILED
- Explanation: The ORB could not establish a listener thread on the port identified by the object reference. The port was already in use or there was an error in creating the daemon thread.
- Minor code: 9
- Static variable: com.sun.rmi.util.MinorCodes.BAD_LOCATE_REQUEST_STATUS
- Explanation: The locator performed a locate request for an object reference and returned a locate reply with a status that is not valid.
- Minor code: 10
- Static variable: com.sun.rmi.util.MinorCodes.STRINGIFY_WRITE_ERROR
- Explanation: An exception was encountered while attempting to create a string-formatted object reference.

**org.omg.CORBA.INV_OBJREF**

- Minor code: 1
- Static variable: com.sun.rmi.util.MinorCodes.NO_PROFILE_PRESENT
- Explanation: The object reference does not contain a profile.
- User response: The current object reference is not valid. Obtain a valid object reference from the object supplier.
- Minor code: 2
- Static variable: com.sun.rmi.util.MinorCodes.BAD_CODE_SET
- Explanation: An unsupported code set or a code set that is not valid was used to write the data to the input stream.
- User response: Use a Unicode or ASCII code set.

**org.omg.CORBA.MARSHAL**

- Minor code: 4
- Static variable: com.sun.rmi.util.MinorCodes.READ_OBJECT_EXCEPTION
- Explanation: An error was encountered while trying to read and convert a marshalled object reference into an in-memory object.
- User response: Ensure that the object (passed as a parameter) is in one of the locations identified by the system CLASSPATH environment variable.
- Minor code: 6
- Static variable: com.sun.rmi.util.MinorCodes.CHARACTER_OUTOFRANGE
- Explanation: While marshalling or unmarshalling an object, a character that is not compliant with ISO Latin-1 (8859.1) was encountered. The character is not in the

range 0 to 255.

**org.omg.CORBA.NO_IMPLEMENT**

- Minor code: 2
- Static variable: com.sun.rmi.util.MinorCodes.GETINTERFACE_NOT_IMPLEMENTED
- Explanation: The get_interface method is not implemented on the server.
- Minor code: 3
- Static variable: com.sun.rmi.util.MinorCodes.SEND_DEFERRED_NOTIMPLEMENTED
- Explanation: Deferred sends are not supported by the ORB.

**org.omg.CORBA.OBJ_ADAPTER**

- Minor code: 1
- Static variable: com.sun.rmi.util.MinorCodes.NO_SERVER_SC_IN_DISPATCH
- Explanation: The object reference could not be dispatached to the server, because an object adapter that matches the object key could not be found.
- User response: Ensure that the object server still services the requested object.
- Minor code: 2
- Static variable: com.sun.rmi.util.MinorCodes.NO_SERVER_SC_IN_LOOKUP
- Explanation: The requested object could not be located, because an object adapter that matches the adapter that matches the object key could not be found.
- User response: Ensure that the object server that processes the locate requests still services the requested object.
- Minor code: 3
- Static variable: com.sun.rmi.util.MinorCodes.NO_SERVER_SC_IN_CREATE_DEFAULT_SERVER
- Explanation: The ORB was unable to create the default object adapter for an object in the server that processes the actual method call.
- Minor code: 4
- Static variable: com.sun.rmi.util.MinorCodes.ORB_CONNECT_ERROR
- Explanation: An error was encountered while trying to connect to an object in the server that processes the actual method call.
- User response:

**org.omg.CORBA.OBJECT_NOT_EXIST**

- Minor code: 1
- Static variable: com.sun.rmi.util.MinorCodes.LOCATE_UNKNOWN_OBJECT
- Explanation: A locate request was performed and the response indicated that the object is not known to the locator.
- User response: Ensure that the locator that processes the locate requests still services the requested object.
- Minor code: 2
- Static variable: com.sun.rmi.util.MinorCodes.BAD_SERVER_ID
- Explanation: The server ID of the server that received the request does not match the server ID of the request object reference. The server that originally served the object is no longer identified by that server ID.
- User response: Obtain a new object reference for the object from the server that is now servicing that object.
- Minor code: 3
- Static variable: com.sun.rmi.util.MinorCodes.BAD_IMPLID
- Explanation: The implementation ID (identified by the object reference) does not match any implementation on the server.
- User response: Obtain a new object reference for the object from the server that is now servicing that object.
- Minor code: 4
- Static variable: com.sun.rmi.util.MinorCodes.BAD_SKELETON
- Explanation: A skeleton that matches the object reference (identified by the object key) could not be found on the server.
- User response:
- Minor code: 5
- Static variable: com.sun.rmi.util.MinorCodes.SERVANT_NOT_FOUND
- Explanation: The object adapter identified by the object key within the object reference could not locate the servant (an object on the server) to process the object request.
- User response: Ensure that the servant is known to the object adapter.

**org.omg.CORBA.UNKNOWN**

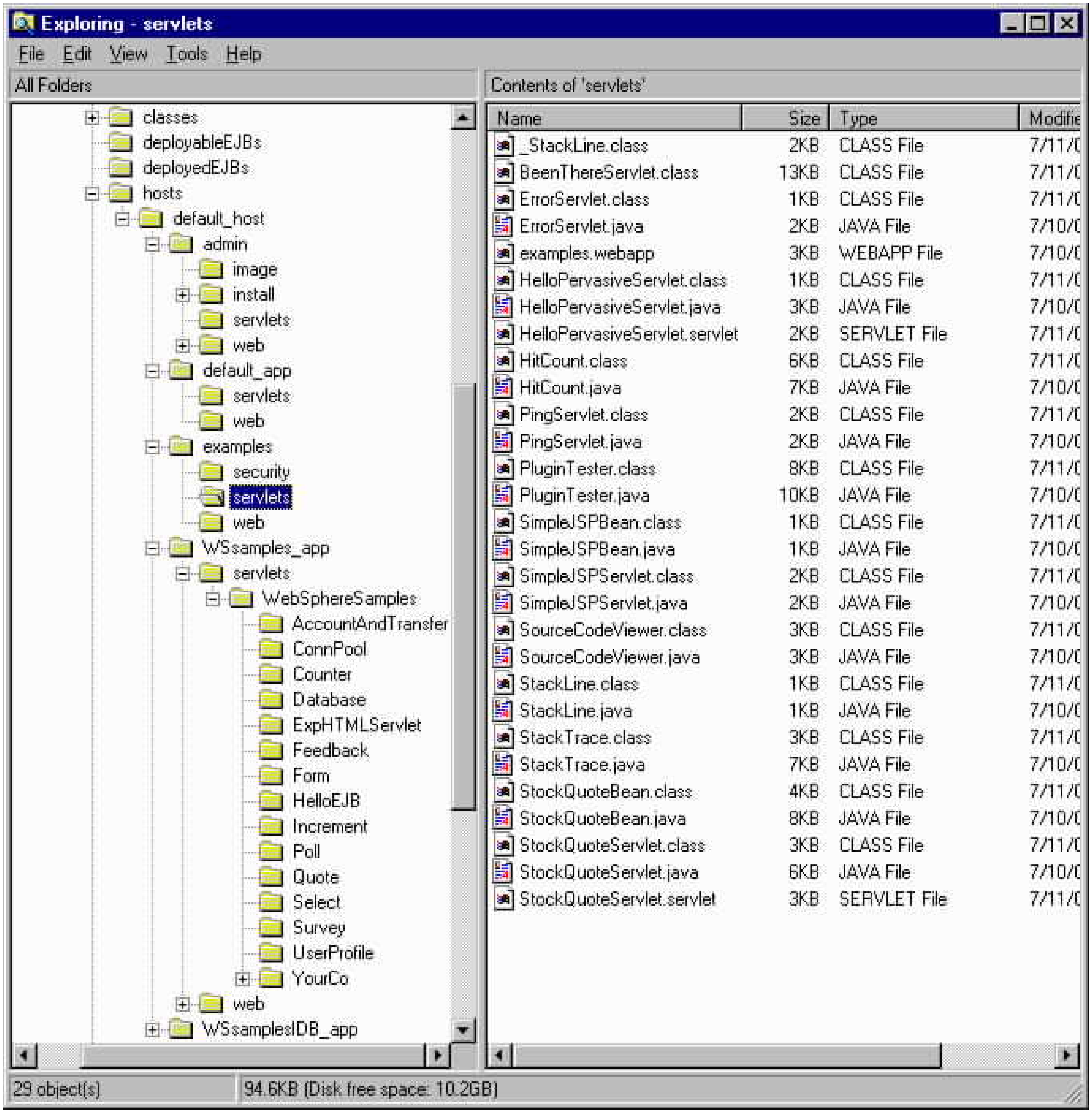
- Minor code: 1
- Static variable: com.sun.rmi.util.MinorCodes.UNKNOWN_CORBA_EXC
- Explanation: The server threw an unknown user exception.
- User response: Ensure that all user exceptions that can be thrown are declared on the throws clause of the method.
- Minor code: 3
- Static variable: com.sun.rmi.util.MinorCodes.RUNTIMEEXCEPTION
- Explanation: The server encountered an unknown application error.
- Minor code: 4
- Static variable: com.sun.rmi.util.MinorCodes.UNKNOWN_SERVER_ERROR
- Explanation: The server threw an unknown exception.

## 8.8: Using internal tools

You can use WebSphere Application Server servlets and internal tools to help diagnose problems.

### Servlets

View file samples.html for information on sample serlvetsshipped with the product. Most WebSphere Application Server servlets are located in the examples directory:



| Name | Size | Type | Modifie |
|---|---|---|---|
| _StackLine.class | 2KB | CLASS File | 7/11/ |
| BeenThereServlet.class | 13KB | CLASS File | 7/11/ |
| ErrorServlet.class | 1KB | CLASS File | 7/11/ |
| ErrorServlet.java | 2KB | JAVA File | 7/10/ |
| examples.webapp | 3KB | WEBAPP File | 7/10/ |
| HelloPervasiveServlet.class | 1KB | CLASS File | 7/11/ |
| HelloPervasiveServlet.java | 3KB | JAVA File | 7/10/ |
| HelloPervasiveServlet.servlet | 2KB | SERVLET File | 7/11/ |
| HitCount.class | 6KB | CLASS File | 7/11/ |
| HitCount.java | 7KB | JAVA File | 7/10/ |
| PingServlet.class | 2KB | CLASS File | 7/11/ |
| PingServlet.java | 2KB | JAVA File | 7/10/ |
| PluginTester.class | 8KB | CLASS File | 7/11/ |
| PluginTester.java | 10KB | JAVA File | 7/10/ |
| SimpleJSPBean.class | 1KB | CLASS File | 7/11/ |
| SimpleJSPBean.java | 1KB | JAVA File | 7/10/ |
| SimpleJSPServlet.class | 2KB | CLASS File | 7/11/ |
| SimpleJSPServlet.java | 2KB | JAVA File | 7/10/ |
| SourceCodeViewer.class | 3KB | CLASS File | 7/11/ |
| SourceCodeViewer.java | 3KB | JAVA File | 7/10/ |
| StackLine.class | 1KB | CLASS File | 7/11/ |
| StackLine.java | 1KB | JAVA File | 7/10/ |
| StackTrace.class | 3KB | CLASS File | 7/11/ |
| StackTrace.java | 7KB | JAVA File | 7/10/ |
| StockQuoteBean.class | 4KB | CLASS File | 7/11/ |
| StockQuoteBean.java | 8KB | JAVA File | 7/10/ |
| StockQuoteServlet.class | 3KB | CLASS File | 7/11/ |
| StockQuoteServlet.java | 6KB | JAVA File | 7/10/ |
| StockQuoteServlet.servlet | 3KB | SERVLET File | 7/11/ |

29 object(s)

94.6KB (Disk free space: 10.2GB)

The following table describes servlets that can be used as debug tools:

| servlet | location | description |
|---|---|---|
| Hit Count | <WebSphere\AppServer>\hosts\default_host\examples\ | Verifies correct implementation of servlets, JSPs, EJBs, and HTTP Session. |
| Snoop servlet | <WebSphere\AppServer>\hosts\default_host\default_app\servlets\Snoop.class | Useful for examining request parameters coming from the client and for verifying the operation of the servlet engine. |
| ShowCfg | <WebSphere\AppServer>\hosts\default_host\examples\ | Useful for validating the configuration of the system. |
| BeenThere | <WebSphere\AppServer>\hosts\default_host\examples\web\beenthere.html | Useful for demonstrating and testing session persistance. |

### Internal tools

The available internal tools apply to specific operations. For example, the jdbctest.java™ tool tests JDK™ settings and database access.

See theWebSphere Problem Determination Tools website for detailed information about these tools. Thewebsite also offers you the opportunity to add ideas about tools and add a new tool. Check the websiteperiodically for updates.

For a quick overview of available logs, traces and tools, see the Problem/Tool/Trace/Log Matrix.

| | |
|---|---|
| Jdbctest.java | Tests JDK™ settings and database connectivity |
| Java™ Name Tree Browser | Displays elements in WebSphere Application Server name space |
| Java™ Socket Level Trace | Describes ORB communication problems over heterogeneous networks via IIOP |
| DrAdmin trace function | Dumps the thread stacks in a server |
| OLT | Object level trace |
| Distributed Debugger | Debugs application level problems |

## 8.8.1: Using the Log Analyzer for Advanced Edition

The Log Analyzer takes one or more activity logs, mergesall of the data, and displays the entries. Based on its "symptomdatabase," it analyzes and interpretsthe error conditions in the log entries to help you debug problems. The Advanced Edition Log Analyzer has a special feature enabling itto download the latest symptom database from the IBM Web site. See"About the Log Analyzer" for details.

- About the Log Analyzer
- About the activity log
- Using the Log Analyzer
- Related tasks:
  - Set the maximum activity.log size
  - Changing the port of the logging service
  - View an activity.log on a remote machine

### About the Log Analyzer

A Log Analyzer quite similar to the one available for use with IBMWebSphere Application Server is available with IBM Component Broker, partof the Enterprise Edition of IBM WebSphere Application Server.

To learn about the Component Broker Log Analyzer, see the Component Brokerproblem determination guide (currently, Chapter 11). You can view thedocument on the IBM Web site without having to obtain Enterprise Edition:
`ftp://ftp.software.ibm.com/software/websphere/info/appserv/v35/ee/cbprbdet.pdf`

The main differences between the Log Analyzer available withWebSphere Application Server and the Component Broker Log Analyzerare the following:

- The Log Analyzer for Advanced Edition is capable of downloading the latest symptom database (bin/symptomdb.xml) from the IBM support site. Use the file -> Update Database -> Adv Symptom Database option in the Log Analyzer interface to take advantage of this feature.
- The functions for ORB trace formatting, minor codes, message IDs and GPF are **not** supported for Advanced Edition
- The script for starting the Log Analyzer is in a different location (see below for instructions)
- The default directory for opening logs for Advanced Edition is the logs directory. For Enterprise Edition, it is the service directory.

### About the activity log

Recall, the Log Analyzer collects messages from various product componentsand places them in a shared file. The file is a binary file located at:
`product_installation_root/logs/activity.log`

The activity.log cannot be easily viewed using a texteditor. The Log Analyzer is the tool for viewing the file.

### Using the Log Analyzer

To view the activity.log using the Log Analyzer:

1. Change directory to:
   `product_installation_root/bin`
2. Run the waslogbr script file, which is called:
   - waslogbr.bat on Windows NT
   - waslogbr.sh on UNIX systems

   It needs to be run from the bin directory cited above.

   This will start the Log Analyzer graphical interface.
3. In the interface:
   1. Select File-> Open.
   2. Navigate to the directory containing the activity.log file.
   3. Select the activity.log file.
   4. Select Open.

### Related tasks

In the course of using the Log Analyzer, you might need to perform the following tasks.

#### Setting the maximum activity.log file size

The activity.log file grows to a predetermined size, then wraps. The default size is 1 Megabyte (MB).

To change the log size:

1. Open the properties file in a text editor:
   `product_installation_root/properties/logging.properties`
2. For the com.ibm.ws.ras.ActivityLogSize property, specify the value you would like, in Kilobytes (KB).

   If an invalid size is entered, the default size is used.

The size change will take effect at the next server startup.

Syntax example:

- To change the log size to 2MB, enter in the line:
  `com.ibm.ws.ras.ActivityLogSize=2048`
  without any spaces in it.

### Changing the port of the logging service

The logging service starts automatically at server startup. It requires theuse of a dedicated port. The default port is 1707.

To change the port value:

1. Stop all application servers and the WebSphere administrative server. (Ifyou do not stop a server, it will not pick up the property changeuntil it is stopped and started again).
2. Open the properties file in a text editor:
   `product_installation_root/properties/logging.properties`
3. For the SHARED_LOG_LOCK_PORT property, specify the value you would like.
4. Start the application and administrative servers that you stopped.

Syntax example:

- To change the port to 1708, specify:
  `SHARED_LOG_LOCK_PORT=1708`

If the port is in use by another application, the logging service might not be able to start or might not function correctly. The activity.log file will not be created or updated correctly. See article 1.2.8 forinformation about how to tell whether a port is currently allocated to anotherapplication.

To diagnose a port conflict, perform these heuristic checks:

- Check to see if the activity.log file has been created, and check the timestamp of the file.
- Check these files:
  `product_installation_root/<server_name>_stderr.logproduct_installation_root/logs/adminserver_stderr.log`

  **Note:** The above paths are the default locations of the files. The administratormight have configured different locations.

  Look for a stack trace such as the following:

```
 java.lang.Exception: Unable to obtain Shared Log Lock on port1707      at
com.ibm.ejs.ras.SharedLogBase.acquireHostLock(SharedLogBase.java:187)        at
com.ibm.ejs.ras.SharedLogWriter.<init>(SharedLogWriter.java:130)     at
com.ibm.ejs.ras.SharedLogWriter.getInstance(SharedLogWriter.java:100)        at
com.ibm.ejs.ras.Tr.initialize(Tr.java:241)   at
com.ibm.ejs.sm.server.ManagedServer.main(ManagedServer.java:121)
```

### Viewing an activity.log file in the absence of a GUI.

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 to the systemon which you are running the WebSphere 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 activity.log file:

1. Change directory to:
   `product_installation_root/bin`
2. Run the showlog tool with no parameters to display the usage instructions:
   - On Windows NT, run showlog.bat
   - On UNIX systems, run showlog.sh

Syntax examples:

- To direct the activity log contents to stdout, use the invocation:
  `showlog activity.log`
- To dump the activity.log to a text file that can beviewed using a text editor, use the invocation:
  `showlog activity.log textFileName`

## 8.9: Thread dumps

This section introduces the concept of thread dumps in WebSphere Application Server.

## What is a thread dump?

A thread represents a work item or task, such as a servlet request. Java processes are usually multi-threaded. This means there can be many tasks occurring simultaneously(i.e. multi-threading)within one JVM (Java Virtual Machine) process. Therefore, understanding what is occurring within a JVM process means obtaining information about all the different threads that are defined within the process.

There are two types of thread dumps that could appear when running Java programs:

- System thread dumps
- Java thread dumps

## System thread dumps

System thread dumps provide a system view of a failing JVM (Java Virtual Machine) process. On Unix systems, they usually appear as core files. On Window's systems they appear as drwtsn32.log files.

System dumps do not understand Java classes. Everything in a system dump is C library oriented. The system dump information provided for JVM processes refers to Java's C libraries and not the referenceclass files.

System dumps should only be interrogated when a Java thread dump is unavailable. Pertinent information can be obtained from system dumps. However, mapping this information back into Java source code is very difficult. The following sections explain how to interrogate the core and drwtsn32.log files. When theyare generated by the system, they need to be interrogated.

### Unix platforms

#### Core files

Core files on Unix systems can be interrogated by dbx and gdb. Dbx is a tool that is part of the AIX install. On Sun, dbx can be installed for an additional expense. The gdb (GNU debugger)is freeware that can be downloaded.

Core file tips:

1. Ensure that the system core file size specification is unlimited.
2. Ensure that the file system containing the core file has enough space.

The following is a sample of how to use ulimit to verify and set the core dump size. If it is too small, a unusable core file will be generated.

**Ulimit** sample:

```
[pwh501]:root> ulimit -a
time(seconds) unlimited
file(blocks) unlimited
data(kbytes) unlimited
stack(kbytes) unlimited
memory(kbytes) unlimited
coredump(blocks) unlimited
nofiles(descriptors) 2000
```

The following commands will change the coredump (-c) and file (-f) to unlimited:

```
ulimit -f unlimited
ulimit -c unlimited
```

The following is an example of using the df command to verify that there isenough room in the file system for the core. The core file is placed in the ./bin directory. On AIX this is in the /usr filesystem. A core file can be 200MB.

**Df** sample:

```
[pwh501]:root> df
Filesystem 512-blocks Free %Used Iused %Iused Mounted on
/dev/hd4 131072 80416 39% 2480 8% /
/dev/hd2 8306688 2835096 66% 76320 8% /usr
/dev/hd9var 606208 55176 91% 390 1% /var
/dev/hd3 475136 459808 4% 32 1% /tmp
```

```
/dev/hd1 1310720 426120 68% 12453 8% /home
/dev/lv00 65536 47048 29% 96 2% /usr/lpp/netviewdm
/dev/lv01 606208 296504 52% 915 2% /db2
/dev/lv02 4014080 2806320 31% 3328 1% /Projects
```

**Note:** These samples were taken from the AIX 4.3.3 system.

## DBX command

The purpose of the dbx command is to provide an environment to debug and run programs under the operating system. The dbx command provides a symbolic debug program for C, C++, Pascal, andFortran programs, allowing you to carry out operations including:

- Examine object and core files
- Provide a controlled environment for running a program
- Set breakpoints at selected statements or run the program one line at a time
- Debug using symbolic variables and display them in their correct format

### DBX syntax

```
dbx [ -a ProcessID ] [ -c CommandFile ] [ -d NestingDepth ] [ -I Directory ]
[-E DebugEnvironment ] [ -k ] [ -u ] [ -F ] [ -r ] [ -x ] [ ObjectFile
[ CoreFile ] ]
```

The ObjectFile parameter is an object (executable) file produced by a compiler. Use the -g (generate symbol table) flag when compiling your program to produce the information the dbx command needs.

**Note:** The -g flag of the cc command should be used when the object file iscompiled. If the -g flag is not used or if symbol references are removedfrom the xcoff file with the strip command, the symbolic capabilities of the dbx command are limited.

If the -c flag is not specified, the dbx command checks for a .dbxinit file inthe user's $HOME directory. It then checks for a .dbxinit file in the user'scurrent directory. If a .dbxinit file exists in the current directory, that file overrides the .dbxinit file in the user's $HOME directory. If a .dbxinit file exists in the user's $HOME directory or current directory, that file's subcommands run at the beginning of the debug session. Use an editor to create a .dbxinit file.

If ObjectFile is not specified, then dbx asks for the name of the object file to be examined. The default is a.out. If the core file exists in the current directory or a core file parameter is specified, then dbx reports the location where the program failed. Variables, registers and memory held in the core image may be examined until execution of ObjectFile begins. At that point the dbx debug program prompts for commands.

**Note:** The commands are referenced in the AIX Version 4.3 Commands Reference, Volume 2.

### DBX tips

The common procedure of interrogating a core file is to change the directory to where the core file resides. You can then issue the command with the binary executable file as the parameter. It is important that the binary executable is used. Usually the java command is a shell script that calls the executable. If you enter the shell script, java, as the parameter a "cannot find" error message is returned.

The following commands show you how to find the binary executable and invoke the dbx command. It also shows an illegal instruction was executed (i.e. Invalid opcode):

----------------------------------------------------------------------------------------

```
[pwh501]:root> cd /usr/jdk_base
[pwh501]:root> find . -name java -print
./bin/aix/native_threads/java
./bin/java
[pwh501]:root> cd /usr/WebSphere/AppServer/bin
[pwh501]:root> ls -l core
-rw-r--r-- 1 root system 191495883 Aug 07 15:08 core
[pwh501]:root> dbx /usr/jdk_base/bin/aix/native_threads/java
Type 'help' for help.
Warning: The core file is truncated. You may need to increase the ulimitfor file and core dump, or free some space on the file system.
Reading symbolic information ...Warning: no source compiled with -g [using memory image in core]
Illegal instruction (reserved addressing fault) in . at 0x0 ($t29)0x00000000 00000001 Invalid opcode.
```

----------------------------------------------------------------------------------------

If you don't know where the java binary is located, the following command will display the true java executable name of the core:

```
strings core | more
```

After you enter dbx, the where command provides a stack trace of where the error occurred. The following example shows a:

- Stack trace
- Output of the help command
- How to exit dbx

**Stack trace**

```
(dbx) where

warning: could not locate trace table from starting address 0x0

ExecuteJava(??, ??) at 0xd2f9913c

do_execute_java_method_vararg(??, ??, ??, ??, ??, ??, ??, ??) at 0xd2fabd30

execute_java_dynamic_method(0x20e355e0, 0x3002fdb0, 0xd3016aa4, 0xd3016aa8, 0x0, 0x0, 0x0, 0x0) at 0xd2fabef4

ThreadRT0(0x3002fdb0) at 0xd300cd88

sysThread_shell(??) at 0xd2fb50a8

pthread._pthread_body(??) at 0xd010f358
```

**Output of the help command**

```
(dbx) help
```

Commands:

```
alias   assign   attribute   call   case   catch
clear   cleari   condition   cont   delete   detach

display(/)   down   dump   edit   file   func

goto   gotoi   help   ignore   list   listi

map   move   multproc   mutex   next   nexti

print   prompt   quit   registers   rerun   return

run   rwlock   screen   search(/?)   set   sh

skip   source   status   step   stepi   stop

stopi   thread   trace   tracei   unalias   unset

up   use   whatis   where   whereis   which
```

Topics:

```
startup   execution   breakpoints   files   data

machine   environment   threads   expressions   scope

set_variables   usage
```

Type "help" for help on a command or topic.

**How to exit dbx**

```
(dbx) quit

[pwh501]:root>
```

Another useful purpose of the dbx command is to monitor a running process. The -a parameter allows the user to attach to a process. The `catch` and `run` commands can be used to walk through the processing of the JVM process and see all signals that are caught. Use of the `help xxx` command will provide additional information on each of the above commands.

**DBXTRACE.SH**

There are shell scripts that call the dbx command and format the thread information from the core file. The name of the script is usually dbxtrace.sh. There is an AIX version and a Solaris version.

Here's a description on how to run the shell script:

```
[pwh501]:root> ./dbxtrace -a
```

Usage: Automate getting dbx trace information

For core files:

Usage: `dbxtrace [executable] [core]` or : `dbxtrace -c corefile`

Example: `dbxtrace /usr/jdk_base/bin/aix/native_threads/java core`

To attach to a running or hung process:

Usage: `dbxtrace -a PID`

Example: `dbxtrace -a 1234`

The following information describes the beginning of the output when using dbxtrace on AIX:

```
[pwh501]:root> ./dbxtrace.sh | more
******************************************
* Failure of this script or dbx may *
* overwrite your existing core file. *
* It is recomended that you rename your *
* existing core file and use the -c flag *
* Do you wish to continue (y/n): ******************************************
Creating subcommand file....
Running dbx...
Type 'help' for help.
warning: The core file is truncated. You may need to increasethe ulimitfor file and coredump, or free some
space on the filesystem.
Reading symbolic information ...warning: no source compiled with -g
```

**Note:** The the user is prompted for (y/n). Therefore, if the user redirects the output to a file `[pwh501]:root>` **`./dbxtrace.sh > myfile 2>&1`** a standalone "y" must be entered before the output is generated.

The output of the dbxtrace.sh provides information about each defined thread. The output has thefollowing sections:

- Error condition
- One line description of each thread
- Detail thread information
- Stack trace of each thread

**Error condition**:
Illegal instruction (reserved addressing fault) in . at 0x0 ($t29)
0x00000000 00000001 Invalid opcode.

**One line description for each thread**:
$t29 is the current thread
thread state-k wchan state-u k-tid mode held scope function
$t1 run blocked 37671 u no sys _pthread_ksleep
$t2 run blocked 38197 u no sys _pthread_ksleep
..
>$t29 run running 46443 k no sys

**Detail thread information**
thread state-k wchan state-u k-tid mode held scope function
>$t29 run running 46443 k no sys
general:
pthread addr = 0x20df04e0 size = 0x18c
vp addr = 0x20e376b4 size = 0x284
thread errno = 2
start pc = 0xf0545994
joinable = yes
pthread_t = 1c1d
scheduler:
kernel =
user = 1 (other)
event :
event = 0x0
cancel = enabled, deferred, not pending
stack storage:
base = 0x20df5738 size = 0x40000
limit = 0x20e35738
sp = 0x20e35040

**Stack trace of each thread**
thread state-k wchan state-u k-tid mode held scope function
*$t29 run running 46443 k no sys
warning: could not locate trace table from starting address 0x0
ExecuteJava(??, ??) at 0xd2f9913c

do_execute_java_method_vararg(??, ??, ??, ??, ??, ??, ??, ??) at 0xd2fabd30
execute_java_dynamic_method(0x20e355e0, 0x3002fdb0, 0xd3016aa4, 0xd3016aa8, 0x0, 0x0, 0x0, 0x0) at 0xd2fabef4

```
ThreadRT0(0x3002fdb0) at 0xd300cd88
sysThread_shell(??) at 0xd2fb50a8
pthread._pthread_body(??) at 0xd010f358
```

### Windows platform

The drwtsn32.log files are similar to core files on Unix. On Windows 2000, these files are found in the following directory: C:\Documents and Settings\All Users\Documents\DrWatson.

After entering drwtsn32 ?, the "Dr. Watson for Windows 2000" box appears. The DrWatson log file overview option will display a screen which explains the format of the drwtsn32.log files. The output of the dbxtrace.sh provides information about each defined thread. The output has the samesection as a Unix platform:

- Error condition
- One line description of each thread
- Detail thread information
- Stack trace of each thread

## Java thread dumps

Java thread dumps provide a Java view of a failing JVM process. Depending on the platform, Java dumps can appear with different names and at different locations.

A Java dump provides information about the executing Java classes and allows the problem determinationprocess to reference the Java source code.

### How to obtain a JAVA Thread Dump

There are two ways to obtain a Java thread dump:

- DrAdmin function
- kill -3 command

DrAdmin works on all platforms. On Unix, the kill -3command serves the same function and is easier to use. Therefore, DrAdmin is discussed in the Windows platform section and kill -3 is discussed in the Unix platforms section.

### Unix platforms

Sometimes Java thread dumps will occur due to an error in the JVM. At other times, the user might need to understand what is occurring within a JVM that is currently active. In either case, the Java thread dump is placed at the location described in the locations table. Information on how to manually obtain a thread dump is available in the remainder of this section.

When a process hangs or is working hard (i.e. looping), it might be helpful to understand what the individual threads of a JVM process are doing. Obtaining a stack trace of the individual threads will provide this information. The kill -3 process ID command provides this stack trace information. This command should not impact the running process.

#### Identifying process IDs

WebSphere supports four processes:

- Nanny - started with startupServer.sh
- Administrative server - started by the nanny process
- Administrative client - started with adminclient.sh
- Managed server - started by the administratiave server either automatically or manually via the administration client console

These processes are usually started in the sequence that they are listed. Therefore, their process IDs increase in value. The ps -ef | grep java command will display all the processes that are associated with java.

The process IDs are listed under the second column in the command's output. The user can not use the administration client interface if managed servers are automatically started by the administrative server.

Unfortunately, the ps -ef | grep java command does not always allow the user to identify the different processes. The command string to start the processes can be very long and the length of the command string saved by the system may not be adequate for the ps -ef | grep java command.

On AIX, the complete command line is listed in the above ps -ef | grep command output. The user can also enter the following commands to focus on an individual process ID:

- ps -ef | grep Nanny
- ps -ef | grep AdminServer
- ps -ef | grep AdminClient
- ps -ef | grep ManagedServer

There could be multiple managed servers running simultaneously. The managed server process ID(s) are also displayed within the ./bin/tracefile with:

```
Starting Server: "Default Server" (pid "116032")
```

Default server is the name of the managed server. On the administration client window, the **General** tab information for the managed server also displays the process ID.

As the root user, the kill -3 *xxxx* can now be entered where *xxxx* is the process ID of the WebSphere JVM in which you need to see a thread dump.

### Location of thread dump

The location of the thread dump depends on the operating system.

| Process | AIX 4.3.3 | Sun OS 5.7 | HP-UX B.11.0.0 |
|---|---|---|---|
| **Administrative server** | ./bin/javacore....txt | Appended to ./logs/tracefile | Appended to ./logs/tracefile |
| **Managed server** | ./bin/javacore...txt | ./Appended to stderr.file for managed server (Note 1) | Appended to stdout file for managed server (Note 1) |
| **Administrative client** | ./bin/javacore...txt | Prompted at window used to enter adminclient.sh | Window used to enter adminclient.sh |
| **Nanny** | ./bin/javacore...txt | Prompted at window used to enter startupServer.sh | Window used to enter startupServer.sh |

**Note 1**: The stderr and stdout files are defined within the managed server configuration. The configuration can be viewed with the administrative console. Click on the managed server (for example default server). The standard output file and standard error file are defined within fields on the **General** Tab.

If the user starts a server in the background, the kill command may not dump the thread information. The workaround for this situation is to do the following:

startupServer.sh &

Ctrl-Z

fg

kill -3 xxxx

### Windows platform

#### DrAdmin.bat file

The DrAdmin.bat file is located in the WebSphere/AppServer/bin directory. The DrAdmin.bat file will execute the DrAdmin function. In Unix, the DrAdmin.bat file is DrAdmin.sh.

#### Find the port number of interest

The next step is to identify the port number for either the administratiave server or a managed server. The port number is different for the administratiave server and each of the managed server(s). The port number values are contained in the standard out files for each of these processes. Information on how to find these files and the port number are described below.

After starting the administratiave server, you should obtain the DrAdmin port number within the .\logs\tracefile file inside the message:

```
DrAdmin available on port xxxx
```

After starting the managed server (for example, default server), you should obtain the DrAdmin port number provided in the standard output file for the managed server. The location of the file can be found with the administrative console interface in the managed server configuration via the **General tab** to standard output field. The message within the file containing the port number is:

```
DrAdmin available on port xxxx
```

#### Execute DrAdmin

The DrAdmin.bat file can now be executed providing the port number obtained above. The format of the command to use is:

```
DrAdmin -serverPort xxxx -dumpThreads
```

where xxxx is the port number from the above message (without the comma).

#### Locate the thread dump

The administratiave server thread information is placed in .\logs\adminserver_stderr.log file. Because this file is not closed, it's length of 0 will not change. The application server thread information is placed in the standard error file which can be found with the administrative console interface in the application server configuration via the **General** tab to standard error field. The thread information is dumped immediately into this file.

In order to view the thread information, copy the above files into a new file. Edit the new file with an HTML editor, which will display the thread information. Some editors (i.e. emacs and vi) will allow you to view the thread information directly from the .\logs\adminserver_stderr.log file or the standard error file.

## How to interrupt a Java thread dump

A thread dump can be forced or can occur when a Java process error occurs. When a thread dump is not forced, it usually means that an error within a Java process has occurred and it should to be investigated. A thread dump of a Java process needs to be forced when the process has a thread deadlock condition. A **thread deadlock condition** is defined as:

*Thread A currently owns Lock X.*
*Thread B owns Lock Y.*
*Thread A is waiting for the release of Lock Y in order to continue processing.*
*Thread B is waiting for the release of Lock X in order to continue processing.*

Because of this stalemate condition, neither thread is able to complete its processing.

**Note:**The referenced Java thread dump information is taken from a sample AIX dump. Java thread dumps on other platforms have similar information, but they may be formatted different.

### Monitors

In order to have a thread safe application, the application may have to ensure that two threads don't execute the same code simultaneously. This can be accomplished with the use of a synchronized()statement or a synchronized modifier of a class method.

Each of the above threads in the thread deadlock condition will create a monitor/lock that will prevent other threads from executing the same code. It is important to understand that threads can be holding multiple monitors/locks while processing a request. Therefore two threads could find themselves in a deadlock condition defined by the following situation:

*Thread A owns Lock X.*
*Thread B owns Lock Y.*
*Thread A is waiting for the release of Lock Y.*
*Thread B is waiting for the release of Lock X.*

Because of this stalemate condition, neither thread is able to complete its processing.

#### Example of a deadlock condition

You can recognize a deadlock when looking within the native stack information. For example, when lookingat the native stack information of Thread A you can easily recognize that it is blocked by a monitor/lock held by Thread B. This information does not appear in the native stack information of the Thread B.Thread B is currently deleting a connection, (deleteConn), from the ConnectionTable. The deleteConn()is a synchronized method which causes a monitor/lock to occur for the ConnectionTable class. There is only one ConnectionTable instance. The monitor/lock held by Thread B is preventing the Thread A process from completing.

The above diagnosis requires an understanding of the involved source code (i.e. which methods are synchronized). However, the Java thread dump does provide the pointers to do this additional investigation.

A summary of the object monitors will provide additional information that identifies Thread A is blocked by Thread B:

```
com.ibm.CORBA.iiop.IIOPConnection@4fe89740: owner: "Thread B""Thread A" (0x36951ba8) blocked
```

Unfortunately, this information does not appear in the summary for the monitor being held by Thread A.

## Stack traces

Stack traces represent the current call path of a thread. Call path information explains what functional calls were made to get to the thread's current location.

### System dump stack trace

Note that the sysAcceptFD() call is the last function called on the stack. It is a system call that was invoked by java_net_PlainSocketImpl_socketAccept() call. The call indicates that a Java thread did an accept operation on a socket. Question marks appear as parameters. This is because the Java process was not run in debug mode. For Java 1.1 installations (i.e. before WebSphere 3.5), debug mode is started by using the java_g command. For Java 1.2 installations(i.e. WebSphere 3.5), the -Xdebug options should be used with the Java command. For either type of installation, the administration console screen for a managed server configuration has a Debug tab. Debug mode can be set within this tab. As stated above, no class file information appears in the stack trace. Only the functions with C libraries are referenced.

### Java dump stack trace

The reader is able to follow the sequence of calls from the run() method through the read() method of the SocketInputSteam class. The package names of the classes are also present. The "Compiled Code" characters appears as parameters in the call because the Java dump occurred for a JVM that was not running in debug mode. When running in debug mode, the line number of the call within the source replaces the "Compiled Code" characters. For Java 1.1 installations (i.e. before WebSphere 3.5), debug mode is started by using the java_g command. For Java 1.2 installations (i.e. WebSphere 3.5), the -Xdebug options should be used with the java command. For either type of installation, the administration console screen for a managed server configuration has a Debug tab. Debug mode can be set with this tab. Another way of obtaining the source line number is to turn off the JIT (Just In Time) compiler. This can be done by starting the JVM with the -Djava.compiler=NONE parameter. This parameter can also be placed on the managed server command line. The command line information can be displayed with the administration console interface on the **General** tab of the managed server configuration.

## WebSphere Application Server thread information

### Object Request Broker information

During startup of the different WebSphere Application Server processes, the processes are initialized and placed in a state to accept additional network activity. One of the first steps in initializing a process is to create an ORB instance. This step will create threads that will be used to complete the initialization step and later accept network activity to be processed.

These activities are described within each of the two diagrams of the next two sections. The diagrams describe:

- Administratiave server startup and immediate administrative server takedown
- Managed server startup with servlet traffic

The administratiave server has two ORBs defined within it. For each ORB is at least one ORB server listener thread that continually waits for input on a port. When input is received, it is dispatched to an ORB server reader thread so the ORB server listener thread can again wait for input on the port. The ORB server reader thread again dispatches the request to a third thread that completes the work activity. The reply to the work activity is sent from the third thread to an ORB client reader thread that receives replies from the ORB reader thread. An ORB request has four steps/threads involved:

1. ORB server listener thread receives input on port X.
2. ORB server reader thread is given a request.

3. Pooled/instantiated thread handles the request and sends a reply.
4. ORB client reader thread handles the reply.

The managed server has one ORB defined within it. There are two ORB server listener threads and multiple ORB client threads. The servlet traffic does not use the ORB for communications. It is done with the plug-in interface. This interface supports a pool of worker threads (Worker#_) that complete the HTTP requests.

The following port numbers are preset:

- 9000 is used for obtaining naming services (i.e. data source names, EJB names)
- 900 is used by the administratiave server to listen for administrative client requests

Other port numbers are randomly chosen for ORB communications.

## Thread names

### ORB threads

The ORB instance creates reader and listener threads. The names of these threads get changed after they begin processing (i.e. during run() method processing). The name is constructed with the following parameters separated by a colon (:):

- ORB information
  - P = unique for this process and algorithmically constructed from a time stamp
  - O = number of ORB's within this process
- Thread type
  - StandardRT = identifies which reader thread is within the ORB
  - CT = client thread
  - LT = listener thread
- Connection values
  - LocalPort = Local port that thread is dealing with
  - RemoteHost = Hostname for ORB server reader thread, or IP address for ORB client reader thread
  - RemotePort = Port number on the remote host for the connection

### Worker#__ (SERVLET ENGINE THREADS)

These thread names begin with Worker# and process HTTP requests.

### Thread-x

These thread names are the default thread name for Windows 2000 and AIX. Because no thread name is provided this name is used. X is incremented as each new thread is created.

### Pooled ORB request dispatch WorkerThread

These threads are created by the main thread (i.e. P=479481:O=0:CT)and handle the request/replies that are sent across IIOP connections.

### Web server plug-in configuration thread

Thread used for setting the Web server configuration.

### Alarm manager

This thread manages the creation of alarm thread x's.

### Alarm thread 1

The alarm thread 1 reclaims unused connections.

### BackgroundLruEvictionStrategy

This thread sweeps a cache, reclaiming the least recently used objects.

### Refresh

This thread insures that any changes to a model get propagated to clones.

## Thread stack traces

When a thread is created, the start() method is used to invoke the run() method. The start() method is executed on one thread and the run() method is executed on the newly created thread. Depending on when the stack trace is obtained, an activity (i.e. piece of work) could have different stack traces. Therefore, thread names have two base method calls. The following text describes these base method calls for the common thread names used for both the administratiave server and the managed server. Two stack trace examples of base method calls are also provided:

**Base method calls**

1. Main or P=xx:O=0:CT
    - run ---> com.ibm.ejs.sm.server.AdminServer.main()

2. ORB server listener thread (JavaIDL Listener or P=xx:O=0:LT=0:port=9000)
    - start ---> com.ibm.ejs.sm.server.AdminServer.main()
    - run ---> com.ibm.CORBA.iiop.ListenerThread.run()
3. ORB server reader thread (JavaIDL Reader for hostname:port# or P=xx:O=0:StandardRT=0:LocalPort=port#:RemoteHost=hostname:RemotePort=port#:)
    - start ---> com.ibm.CORBA.iiop.ListenerThread.run()
    - run ---> com.ibm.CORBA.iiop.StandardReaderThread.run()
4. ORB client reader thread (JavaIDL Reader for ipaddr:port# or P=xx:O=1:StandardRT=1:LocalPort=port#:RemoteHost=ipaddr:RemotePort=port#:)
    - start ---> com.ibm.ejs.sm.server.AdminServer.main()
    - run ---> com.ibm.CORBA.iiop.StandardReaderThread.run()
5. Pooled ORB request dispatch WorkerThread
    - start ---> com.ibm.CORBA.iiop.StandardReaderThread.run()
    - run ---> com.ibm.ejs.oa.pool.ThreadPool$PooledThread.run()
6. Worker#__
    - start ---> com.ibm.ejs.sm.server.AdminServer.main()
    - run ---> com.ibm.servlet.engine.oselistener.outofproc.OutOfProcThread$CtlRunnable.run() java.lang.Thread.run()
7. Web server plug-in configuration thread
    - start ---> com.ibm.ejs.sm.server.AdminServer.main()
    - run ---> com.ibm.servlet.engine.oselistener.outofproc.OutOfProcThread$CtlRunnable.run() java.lang.Thread.run()
8. Alarm manager
    - start ---> com.ibm.ejs.sm.server.AdminServer.main() <--AdminServer
    - com.ibm.ejs.oa.pool.ThreadPool$PooledThread.run() <--AppServer
    - run ---> com.ibm.ejs.util.am.AlarmManagerThread.run() java.lang.Thread.run()
9. Alarm thread 1
    - start ---> com.ibm.ejs.util.am.AlarmManagerThread.run() java.lang.Thread.run() <--AdminServer
    - com.ibm.ejs.oa.pool.ThreadPool$PooledThread.run() <--AppServer
    - run ---> com.ibm.ejs.oa.pool.ThreadPool$PooledThread.run() <--AdminServer
    - com.ibm.ejs.util.am.AlarmThread.run() <--AppServer
10. BackgroundLruEvictionStrategy
    - start ---> com.ibm.ejs.sm.server.AdminServer.main()
    - run ---> com.ibm.ejs.util.cache.BackgroundLruEvictionStrategy.run()
11. RefreshThread
    - start ---> com.ibm.ejs.sm.server.AdminServer.main()
    - run ---> com.ibm.ejs.wlm.server.config.ServerGroupRefresh$RefreshThread.run()

**Examples**

Thread dump of a standard reader thread:

```
"P=863240:O=1:StandardRT=16:LocalPort=10502:RemoteHost=gofast:RemotePort=2619:"
(TID:0x11ccef0, sys_thread_t:0xcdd81d0, state:R, native ID:0x128) prio=5
>at java.net.SocketInputStream.socketRead(Native Method)
at java.net.SocketInputStream.read(SocketInputStream.java(Compiled Code))
at com.ibm.rmi.iiop.Message.readFully(Message.java(Compiled Code))
at com.ibm.rmi.iiop.Message.createFromStream(Message.java:173)
at com.ibm.CORBA.iiop.IIOPConnection.createInputStream(Unknown Source)
at com.ibm.CORBA.iiop.StandardReaderThread.run(Unknown Source)
```

The base method, com.ibm.CORBA.iiop.StandardReaderThread.run(), is identified as the run base method for JavaIDL Reader for hostname:port# threads. Also, the thread is waiting for input because it is in the java.net.SocketInputStream.socketRead() method.

Thread dump of a worker thread:

```
"Worker#49" (TID:0x10793660, sys_thread_t:0xab25b0, state:R, native ID:0x19a) prio=5
at com.ibm.servlet.engine.oselistener.outofproc.NativeServerQueueImp.nativeGetSeviceMessageId()
at com.ibm.servlet.engine.oselistener.outofproc.NativeServerQueueImp.getSeviceMessageId()
at
com.ibm.servlet.engine.oselistener.serverqueue.SQWrapperEventSource$SelectRunnable.getNewConnectionFromQueue()
at com.ibm.servlet.engine.oselistener.serverqueue.SQWrapperEventSource$SelectRunnable.run()
at com.ibm.servlet.engine.oselistener.outofproc.OutOfProcThread$CtlRunnable.run()
at java.lang.Thread.run()
```

The base method, java.lang.Thread.run(), is identified as the run base method for Worker#__ threads. Also, the thread is waiting for input from the Web server plug-in (native code) because it is in the com.ibm.servlet.engine.oselistener.outofproc.NativeServerQueueImp.nativeGetSeviceMessageId() method.

## Administrative Server Startup with Immediate Takedown Diagram

The following diagram has highlighted request flows that start with a SendReqXXX where XXX is the port number of the send request. The steps in the flow changes between different threads. The sequence of the steps are identified with, for example, 1A,1B, 1C and 1D. It also shows how the port that the request is sent to determines which thread the proccessing has completed.

### Diagram Legend

In each diagram, every continuous line (-------) is a thread. The name of the thread always appears between (...). The letters in the diagram have the following meanings:

C = Thread name changed to (.....)
S = Start method called on this thread
R = Run method called on this thread
W = Thread is in wait state waiting for notify
WM = Thread is waiting for message from plugin (Worker# threads only)
SendReq____ = Request sent to port number (____)
SendReply___ = Reply sent to port number (____)

For example:

C(P=479481:O=0:CT) = thread name is changed to P=479481:O=0:CT
R = thread is placed in a running state

### Diagram

ORB 0 Threads(i.e. O=0)

main

|

|

C(P=479481:O=0:CT)

|

|S(JavaIDL Listener) R C(P=479481:O=0:LT=0:port=9000)

|-------------------------------------------------------------------------------------------> | |

| S(JavaIDL Reader for rbostick:1294)

| |

| |

| R

| |

| |

| C(P=479481:O=0:StandardRT=0:LocalPort=9000:

| RemoteHost=rbostick:RemotePort=1294:)

| |

| **1B**

| |S(Thread-1) R

SendReq9000(**1A**) |-**1C**--------------------->

| |

| **2B**

| |S(Thread-2) R

SednReq9000(**2A**) |-**2C**--------------------->

| |

| **5B**

| |

| |

| |

| V

|

```
|S(JavaIDL Reader for 9.27.63.245:9000) R C(P=479481:O=1:StandardRT=1:LocalPort=1294:
| RemoteHost=9.27.63.245:RemotePort=9000:)
|------1D--2D--5D--------------------------------------------------------------------------> |
|
|
|
|
|
|
|
|
|
|
|
V
ORB 1 Threads (i.e. O=1)
|
|
|
|S(JavaIDL Listener) R C(P=479481:O=1:LT=1:port=1295)
|-----------------------------------------------------------------------------------------> |
|
|
|
|
|S(JavaIDL Listener) R C(P=479481:O=1:LT=2:port=1296)
|----------------------------------------------------------------------------------------->
| |
| S(JavaIDL Reader for rbostick:1299)
| |
| R
| |
| C(P=479481:O=1:StandardRT=5:LocalPort=1296:
| RemoteHost=rbostick:RemotePort=1299:)
| |
| 6B
| |
| V
|
|
|
|S(JavaIDL Reader for 9.27.63.245:1299) R C(P=479481:O=1:StandardRT=4:LocalPort=1299:
| RemoteHost=9.27.63.245:RemotePort=1296:)
|------6D--------------------------------------------------------------------------------->
|
|
|
```

|S(JavaIDL Listener) R C(P=479481:O=1:LT=3:port=900)

|-------------------------------------------------------------------------------------------> | |

| S(JavaIDL Reader for rbostick:1297)

| |

| R

| |

| C(P=479481:O=1:StandardRT=3:LocalPort=900

| :RemoteHost=rbostick:RemotePort=1297:)

| |

**| 3B**

| |

**| 4B**

| |

| V

|

|

|S(JavaIDL Reader for 9.27.63.245:900) R C(P=479481:O=1:StandardRT=2:LocalPort=1297:

| RemoteHost=9.27.63.245:RemotePort=900:)

|-----**3D**--**4D**---------------------------------------------------------------------------------->

|

|

|

SendReq900(**3A**)

|

SendReq900(**4A**)

|

SendReq9000(**5A**)

|

SendReq1296(**6A**)

|

SednReq1296(**7A**)

|

Other Threads

|

|

|

|

|

|S(Pooled ORB request dispatch WorkerThread) W R

|------**3C**--**5C**---------------------------------------------------------------------------->

|

|

|

|S(Pooled ORB request dispatch WorkerThread) W R |------**4C**--**6C**---------------------------------------------------------------------------->

|

|

|

```
|
|S(Alarm Manager) R
|------------------------------------------------------------------------------------------->
| S(Alarm Thread 1)
| |
| R
| |
| V
|
|S(Thread-3) R
|------------------------------------------------------------------------------------------>
|
|
|S(Thread-4) R
|------------------------------------------------------------------------------------------>
| | | | | |
| S(Thread-8) S(Thread-9) S(Thread-10) S(Thread-11) S(Thread-12) | | | | | |
| R R R R R
| | | | | |
| V V V V V
|
|
|S(Worker#0) R S(Worker#0) R
|---------------------------------------------------------------------------------------------->
|
|
|S(WebServer-Plugin-Cfg-Thread) R
|---------------------------------------------------------------------------------------------->
|
|S(BackgroundLruEvictionStrategy) R
|---------------------------------------------------------------------------------------------->
|
|S(RefreshThread) R
|----------------------------------------------------------------------------------------->
V
```

**Thread-1(2) (worker threads)**

- start ---> com.ibm.CORBA.iiop.StandardReaderThread.run()run ---> com.ibm.CORBA.iiop.WorkerThread.run()

**Thread-3 (transaction timeout)**

- start ---> com.ibm.ejs.sm.server.AdminServer.main()
- run ---> com.ibm.ejs.jts.tran.JavaClock.run()

**Thread-4 (used for administrative server takedown)**

- start ---> com.ibm.ejs.sm.server.AdminServer.main()
- run ---> com.ibm.ejs.sm.server.ManagedServer$DiagonisticThread.run()

**Thread-8,9,10,11,12 (threads for takedown process)**

- start ---> com.ibm.ejs.sm.server.ManagedServer$DiagonisticThread.run()
- run ---> com.ibm.ejs.sm.util.task.AsyncTaskEngine$WorkerThread.run()

**Note:** Thread-x are default names of threads. The above numbers may be different depending on the system that the administratiave server runs on.

## Managed Server Startup with Servlet Traffic Diagram

The Worker#_threads are the threads on which servlet requests are processed. The threads start during the managed server startup and wait on input from the Web server plug-in interface.

main

|

|

C(P=905990:O=0:CT)

|

|

|S(Thread-0) R

|------------------------------------------------------------------------------------------>

|

|

|S(Pooled ORB request dispatch WorkerThread) W R

|----------------------------------------------------------------------------------------------->

| | | | | | | |

| S | | S(**Worker#0**)S(**Worker#1**).......S(**Worker#24**) S(Thread-6) (BackgroundLruEvictionStrategy) | | | |

| | | R WM WM WM R

| R | | | | | |

| | S(AlarmManager) | S(**Worker#0**) | **Servlet** |

| | | | | | **Request** |

| | | S(pluginRegenScheduler) | | | |

| | R | WM | WM |

| | | V V V V V

| | |

| | |

| V |S(AlarmThread1) R

| |---------------------------->

| V

|

|

|S(Pooled ORB request dispatch WorkerThread) W R W |------------------------------------------------------------------------------------->

|

|S(Thread-1) R

|--------------------------------------------------------------------------------------->

|

|

|

|S(Thread-3) R

|--------------------------------------------------------------------------------------->

|

|

|

|

**ORB 0 Threads (i.e. O=0)**

|

|S(JavaIDL Reader for 9.27.63.129:9000) R C(P=905990:O=0:StandardRT=0:LocalPort=1480:

```
| RemoteHost=9.27.63.129:RemotePort=9000:
|------------------------------------------------------------------------------------>
|
|
|
|S(JavaIDL Listener) R C(P=905990:O=0:LT=0:port=1481)
|------------------------------------------------------------------------------------->
|
|
|S(JavaIDL Reader for 9.27.63.129:1434) R C(P=905990:O=0:StandardRT=1:LocalPort=1482:
| RemoteHost=9.27.63.129:RemotePort=1434:)
|------------------------------------------------------------------------------------->
|
|
|
|
|S(JavaIDL Reader for 9.27.63.129:900) R C(P=905990:O=0:StandardRT=2:LocalPort=1483:
| RemoteHost=9.27.63.129:RemotePort=900:)
|------------------------------------------------------------------------------------->
|
|
|
|S(JavaIDL Reader for 9.27.63.129:1433) R C(P=905990:O=0:StandardRT=3:LocalPort=1484:
| RemoteHost=9.27.63.129:RemotePort=1433:)
|------------------------------------------------------------------------------------->
|
|
|
|
|S(JavaIDL Listener) R C(P=905990:O=0:LT=1:port=1485)
|------------------------------------------------------------------------------------------>
| |
| S(JavaIDL Reader for rbostick:1487)
| |
| R
| |
| C(P=905990:O=0:StandardRT=4:LocalPort=1485:
| RemoteHost=rbostick:RemotePort=1487:)
| |
| |
| V
V
```

**Thread-0 (transaction timeout)**

- start ---> com.ibm.ejs.sm.server.ManagedServer.main()
- run ---> com.ibm.ejs.jts.tran.JavaClock.run()

**Thread-1 (Used for logging messages)**

- start ---> com.ibm.ejs.sm.server.ManagedServer.main()

- run ---> com.ibm.ejs.sm.server.SeriousEventListener$DeliveryThread.run()

### Thread-3

>

- start ---> com.ibm.ejs.sm.server.ManagedServer.main()
- run ---> com.ibm.ejs.sm.server.ManagedServer.main()

### Thread-6 (administrative server ping)

- start ---> com.ibm.ejs.oa.pool.ThreadPool$PooledThread.run()
- run ---> com.ibm.ejs.sm.server.ManagedServer$PingThread.run()

**Note:** Thread-x are default names of threads. The above numbers may be different depending on the system the managed server runs on.

## Summary

In multi-processing and multi-thread environments, problem determination can require analysis of actively running threads. This thread information can be obtained with system thread dumps and Java thread dumps. When doing problem determination in a WebSphere Application Server environment, Java thread dumps provide much more information and are recommended. However, sometimes system thread dumps are the only information obtained and should be interrogated.

When dealing with thread deadlock problems, Java thread dumps can be forced using kill -3 on Unixplatforms and DrAdmin on all platforms.

The output of these commands provides thread information necessary to diagnose the problem.

## 8.10: Applying e-fixes

E-fixes are individual fixes for critical problems. They have been individually tested,but not integration tested and should only be applied if you have a critical problem without a valid workaround. They may be applied to both versions of WebSphere, except where specifically noted. All e-fixes are rolled into the next scheduled FixPack. Each fix has a readme file with installation instructions.

To learn about the fixes made available since the last FixPack, see the FixPacks and E-fixes website.

# 8.11: Resource reference

Use these links to learn about other performance tools and techniques.

- 9.1: Tuning the product

# 8.12: Various problem determination topics

This section will provide information about various problem determination topics.

# 8.13: Problem determination hints and tips

When you encounter an error or problem with WebSphere Application Server, you can followthe Hints and Tips to help you quickly gather relevant data to diagnose the problem.

The referenced link provides access to the WebSphere Application Server Standardand Advanced technotes. To view version 3.5 specific technotes:

1. Go to the navigation frame located on the left.
2. Enter 3.5 in the **search** box.
3. Select *Just this category* from the pull-down menu.
4. Press ***Go***.
5. The V3.5 technotes appear in the same window.

# 8.14: How to report a problem to IBM

Use the information in this section to help you report a problem to IBM.

Before reporting problems to IBM, please review the known problems in the Release Notes,Hints and Tips, FAQ's, and other resources on the support website. If you find that the problem is not a known defect, then report the problem to IBM.

There are a variety of ways to report your problem to IBM:

- Phone
- Fax
- Internet

If you need assistance with problems, you are required to purchase technical support. You can select the exact mix of services to fit your specific business needs. IBM Software Support is delivered in a consistent manner for all IBM software products based upon the way in which a product is charged (one time charge or monthly license charge basis).

You can report suspected defects via fax, mail or electronically until the product's service expiration date. This free service is called Warranty/Defect Support. For information on reporting suspected defects, call 1-800-237-5511 in theUnited States and Puerto Rico. In Canada, call 1-800-465-9600. Telephone numbers for countries outside North Americaare also available. The service expiration date is defined in your License Information booklet under Program Services.

## What to provide when reporting problems

You will need the following information available when reporting a problem to IBM:

- The product name and version number
- The kind of hardware and software you are using
- What happened and what you were doing when the problem occurred
- Whether you tried to solve the problem and how
- The exact wording of any messages displayed

After you have reported a problem to IBM support using any of the methods above, especially by phone, you might want to provide relevant logs, traces or files. You can also send an ASCII text description of the problem in your own words. Send logs and text files together in a zip file for ease of transfer.

Follow these steps to send files to IBM:

1. Note the problem record number assigned to you by IBM support.
2. FTP testcase.software.ibm.com
3. Login: anonymous
4. Password: [your email id]
5. Change directory: cd /ps/toibm/internet
6. Make a directory: mkdir pmrnumber [use your problem number, for example, pmr89401]
7. Put [filename]
8. Call IBM Support back and ask that it be noted in your problem record that files are available on the testcase ftp server. Give the path to the files. Files will remain available on the testcase ftp server for 72 hours and will then be deleted.

## Technical support by phone

If you are a licensed customer in the U.S. or Puerto Rico who has a support contract and youneed support, please call IBM Support at 1-800-237-5511. In Canada, call 1-800-IBM-SERV (1-800-426-7378). Telephonenumbers for countries outside North America are also available.

If you are a licensed customer and wish to purchase support, you may contact IBM or yourIBM authorized business partner.

If you have an IBM customer number, call 1-888-426-4343 Monday - Friday 8:00 a.m. to 7:00 p.m. Eastern Standard Time.
In Canada, call 1-800-465-9600 Monday - Friday 8:00 a.m. to 5:00 p.m. Central Standard Time.

If you do not have an IBM customer number, call 1-800-237-5511 Monday - Friday 8:00 a.m. to 5:00 p.m. Central Standard Time.
In Canada, call 1-800-465-9600 Monday - Friday 8:00 a.m. to 5:00 p.m. Central Standard Time.

## Technical support by fax

Contact us via the Faxback System: 1-800-426-4329.
Telephone numbers for countries outside North America are also available.

## Technical support on the Internet

Online help is available through the IBM Support Line.Support Line is the service offering through which IBM delivers electronic support for installation, usage, andcode-related questions. Electronic support is also available through Passport Advantage's online incident report page. Solution developers can also receive online help through the PartnerWorld for Developers.

Information on IBM SupportLine and IBM Services is available on the Internet at the URL listed above. For IBM Lotus Passport Advantage customers, support information is also available at this Internet site.

**Note:** Information may not apply to all products. Support information is subject to change without notice.

# Contents

**Figures**

**Tables**

**Using the JRas Message Logging and Trace Facility**

- Introduction
  - Overview of messages and trace
- The WebSphere JRas programming model
  - Naming and managing loggers
  - Message and trace event types
- Using JRas loggers
  - Creating resource bundles and message files
  - Creating manager and logger instances
  - Using loggers

## Introduction

The IBM(R) JRas toolkit is a set of Java(TM) packages that enablesdevelopers to incorporate message logging and trace facilities into Javaapplications. Although JRas is a standalone product, it has beencustomized for use with the Standard, Advanced, and Enterprise (ComponentBroker) Editions of WebSphere(TM) Application Server. The WebSphereimplementation of JRas integrates with the WebSphere run-time environment andsystem-management utilities (for instance, Advanced Edition'sAdministrative Console and Component Broker's System Manager).This document discusses the WebSphere implementation of JRas and using it towrite WebSphere applications that log and manage application-specific messagesand trace. Use of the non-WebSphere implementation of JRas is notdiscussed in this document.

**Note:**

> The non-WebSphere (base) implementation of JRas is not supported for use withWebSphere Application Server. The use of JRas with WebSphere issupported only with the WebSphere-specific JRas implementation and programmingmodel discussed in this document.

### Overview of messages and trace

Applications often need to provide information about their internaloperations to users, system administrators, programmers, and other interestedparties. This information is typically provided as text that can besent to a console or terminal, written to a log file, directed to a standardoutput or error device, or all three. The JRas toolkit dividesinformational text into the following two categories:

- *Messages*, consisting of information about the application thatis brief, clear, and meaningful to an end user. An example of a messageis a string indicating that the application started successfully.Messages are generated by default; they are not normallysuppressed. Messages can be localized; that is, the messagecatalogs can be translated into various national language versions, andmessages can be displayed in the user's preferred language.
- *Trace*, consisting of detailed technical information about thecurrent state of one or more of the application's internal datastructures, including summaries of all objects in those datastructures. Trace information is meant for use by developers andsupport personnel when debugging applications; it is not generallyintended for use by end users. An example of trace information is astring listing an error, the time at which the error occurred, the thread inwhich the error occurred, the method that was being executed when the erroroccurred, and a description of the error. Trace information is notnormally generated by applications and is enabled only to help resolvespecific problems, because the creation of trace information consumes systemresources beyond the application's normal requirements. Trace isnot localizable; that is, it cannot be translated into national languageversions.

The JRas packages implement objects called *loggers*,*handlers*, *formatters*, and *managers* to providemessaging and trace capabilities. These objects are described in thefollowing list.

- *Loggers* are the primary objects with which the application codeinteracts.
- *Handlers* receive data that is to be logged from alogger.
- *Formatters* are objects invoked by handlers to formatdata.
- *Managers* provide methods to predefine and manage logger,handler, and formatter configurations. These configurations can be keptin a persistent data store. Using managers simplifies programming withJRas; when a manager is used to obtain a logger, the manager retrievesthe logger's configuration data, creates the logger and populates it withthe correct handlers, performs any other needed tasks, and returns theconfigured logger to the caller. The Manager class provided withWebSphere is WebSphere specific and cannot be used with generic JRasimplementations. Using this class to create and manage WebSphere JRasobjects ensures that all derived objects (loggers, handlers, and formatters)conform to the requirements of the WebSphere JRas implementation.

To view message and trace text, you must read the appropriate logfiles. WebSphere currently logs all messages to single-level logfiles; that is, application messages and run-time messages are written tothe same log file. It is recommended that you monitor the size of thelog files and increase the allowable size of the files depending on the numberof messages written to the log. WebSphere also logs all trace events,whether application trace or run-time trace, to the same trace logfile. All editions of WebSphere Application Server provide facilitiesto view message and trace logs; see the documentation for your edition ofWebSphere for more information.

## The WebSphere JRas programming model

This section discusses the supported model for programming with JRas inWebSphere Application Server.

In WebSphere, you create and manage JRas loggers and managers by using theManager class of the com.ibm.websphere.raspackage. The Manager class provides mechanisms to obtain JRas messageand trace loggers that are integrated with WebSphere; it also providesthe ability to group trace loggers into logical groups. The basicprocess for creating JRas objects is to retrieve a reference to the JRasmanager by using the getManager method of thecom.ibm.websphere.ras.Manager class, then toretrieve message and trace loggers by using methods on the returnedmanager. See Creating manager and logger instances for sample code illustrating this process.

The retrieved loggers are implementations of the RASIMessageLogger andRASITraceLogger interfaces. You then program to these interfaces, bothof which are derived from the RASILogger interface. The loggers arestateful objects with their states tied to an existing Java Virtual Machine(JVM) and run-time instance. These interfaces are discussed in Using loggers.

**Note:**

> Although loggers implement the Java java.io.Serializableinterface, they must not be serialized.

## Naming and managing loggers

This section discusses considerations for naming and managingloggers.

WebSphere JRas loggers have no predefined granularity or scope. Anapplication consisting of many different classes can be instrumented by usinga single logger, can be subdivided into several components with a logger foreach component, or can have a logger for each class.

Loggers are named objects; the manager maintains a hierarchical namespace of loggers, with separate name spaces for message loggers and traceloggers. For each unique logger name, the logger instance is created onthe first request to the manager and the same instance is returned onsubsequent calls. The following recommendations apply to namingloggers:

- To prevent name-space conflicts, it is recommended that a dot-separated,fully qualified class name be used to name each logger.
- It is recommended that the full logger name reflect the name of the classthat retrieves the logger from the manager.
- Application developers are responsible for ensuring that the logger namesused by an application do not conflict with names in use by the WebSphere runtime; using full logger names based on retrieval class namesautomatically provides this assurance.
- Because of potential name-space conflicts and limitations in the size ofthe name space, it is recommended that any given class have no more than onemessage logger or trace logger associated with it.
- The name `ORBRas` is reserved for use by the WebSphere runtime. Do not use this name in WebSphere applications that useJRas.

The WebSphere run time and system-management utilities enable you to enableand disable trace at any level of the name-space hierarchy. Changingthe trace state at any level of the hierarchy automatically makes the samestate change for all child levels. For instance, enabling trace at themiddle level of a hierarchy automatically enables trace for all levels belowthe middle level.

Trace loggers can be combined into logical sets called *groups* totrack events across various components of an application. For example,if an application contains three different components, you can create a groupthat includes trace loggers from each component, thereby providing a way totrace the flow of a particular function across all three components.Application developers must provide group names that are unique to theapplication and that do not conflict with other group names in the name space,including names used by the WebSphere run time.

JRas objects are managed by the WebSphere run time. When a logger iscreated, the JRas manager queries the WebSphere system-management utility todetermine the initial state for the logger's mask. The state ofthe mask is updated dynamically in accordance with settings provided to thesystem-management utility. The default initial states for the differenttypes of loggers are as follows:

- For message loggers, the default initial state is always for logging to beenabled to the logger's specified state. There is currently no wayto specify an initial state of disabled. For a list of possible initialstates, see Table 1.
- For trace loggers, the default initial state is for logging to bedisabled; however, an initial state of enabled can be specified by usingthe appropriate WebSphere system-management utility. The tracelogger's mask is set as specified in the system-managementutility. For a list of possible initial states, see Table 2 and Table 3. Some editions of WebSphereApplication Server enable you to change the state of the mask dynamically byenabling tracing for one or more trace loggers; refer to thedocumentation for your WebSphere system-management utility for moreinformation.

  All enabling and disabling of trace must be performed through theappropriate WebSphere system-management utility.

### Message and trace event types

This section discusses the message and trace types that are availablethrough the WebSphere implementation of JRas. Message types areprovided by the RASIMessageEvent interface, and trace types are provided bythe RASITraceEvent interface.

#### Message types and usage

Message types are provided by the RASIMessageEvent interface. Typesinclude the following:

- `TYPE_INFORMATIONAL` for informational messages. This typecan be abbreviated as `TYPE_INFO`.
- `TYPE_WARNING` for warning messages. This type can beabbreviated as `TYPE_WARN`.
- `TYPE_ERROR` for error messages. This type can beabbreviated as `TYPE_ERR`.

These types, which are provided by JRas, do not correspond exactly to themessage types supported by the different editions of the WebSphere runtime. The following table shows the mappings between the JRas messagetypes and their WebSphere equivalents. Note that the Enterprise Editiontypes apply to Component Broker on workstations.

Table 1. JRas message types and their WebSphere equivalents

| JRas message type | Equivalent WebSphere Standard/Advanced Edition type | Equivalent WebSphere Enterprise Edition (Component Broker forworkstations) type |
|---|---|---|
| `TYPE_INFO`,<br>`TYPE_INFORMATION` | `Audit` | `Informational` |
| `TYPE_WARN`,<br>`TYPE_WARNING` | `Warning` | `Warning` |
| `TYPE_ERR`,<br>`TYPE_ERROR` | `Error` | `Error` |

#### Trace types and usage

Trace types are provided by the RASITraceEvent interface. Thisinterface defines two sets of JRas trace types: a basic set of leveledtypes for simple trace implementations and a more complex set of nonleveledtypes that can be logically combined to create precise information about anygiven trace event. It is recommended that only one of these sets beused in any given application.

The basic set of types consists of the `TYPE_LEVEL1`,`TYPE_LEVEL2`, and `TYPE_LEVEL3` trace levels. Theselevels are hierarchical; enabling a higher level of trace automaticallyenables all levels beneath it (for instance, enabling `TYPE_LEVEL2`automatically enables `TYPE_LEVEL1`).

The complex set of types consists of the following trace values:

```
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

TYPE_SVC
```

These values can be combined logically (that is, by using operators suchas AND, OR, and NOR) to provide detailed information about any given traceevent.

As with the message types, the JRas trace types do not correspond exactlyto the types used by the WebSphere run time. The following tables showthe mappings between the JRas trace types and their WebSphereequivalents. Note that the WebSphere equivalents apply to StandardEdition, Advanced Edition, and, for Enterprise Edition, Component Broker onworkstations.

**Table 2. Leveled JRas trace types and their WebSphere equivalents**

| **JRas level event type** | **WebSphere equivalent** |
|---|---|
| `TYPE_LEVEL1` | `Event` |
| `TYPE_LEVEL2` | `Entry/Exit` |
| `TYPE_LEVEL3` | `Debug` |

### Table 3. Nonleveled JRas trace types and their WebSphere equivalents

| **JRas nonleveled event types** | **WebSphere equivalent** |
|---|---|
| `TYPE_ERROR_EXC`,<br>`TYPE_OBJ_CREATE`,`TYPE_OBJ_DELETE`,<br>`TYPE_SVC` | `Event` |
| `TYPE_API`,<br>`TYPE_CALLBACK`,`TYPE_ENTRY_EXIT`,<br>`TYPE_PRIVATE`,<br>`TYPE_PUBLIC`,`TYPE_STATIC` | `Entry/Exit` |
| `TYPE_MISC_DATA` | `Debug` |

## Using JRas loggers

This section discusses how to use JRas loggers in WebSphereapplications. Creating resource bundles and message files provides an overview of creating resourcebundles to provide localized (translated) messages. Creating manager and logger instances discusses how to obtain a JRas manager, and subsequently howto obtain message and trace loggers. Using loggers describes the logger interfaces and shows how to usethem.

### Creating resource bundles and message files

This section provides an overview of how to create resource bundles thatcan be translated to provide localized messages in WebSphereapplications. The Java programming language provides thejava.util.ResourceBundle class and its subclasses,java.util.ListResourceBundle andjava.util.PropertyResourceBundle, to enable national languagesupport for applications. The ResourceBundle class is used inconjunction with the java.text.MessageFormat class to providelocalized (translated) text support. See the Java documentation for afull discussion of the ResourceBundle and MessageFormat classes.

ResourceBundle is a class that encapsulates the retrieval of text.Entries in a resource bundle consist of message keys and their correspondingmessage text. When a resource bundle is translated, only the messagetext is translated into the national language. The translated resourcebundles are packaged together and shipped with the application to providelocalized messages.

This section discusses how to create resource bundles in the form of textproperties files that can be accessed by PropertyResourceBundle. Youcan also create resource bundles by using a Java class that extendsListResourceBundle. The class encapsulates the mapping of keys tovalues by using arrays. For information on creating resource bundles byusing ListResourceBundle, see the Java documentation.

The simplest way to create a resource bundle is to create a text propertiesfile that lists message keys and the corresponding messages. Theproperties file must have the following characteristics:

- Each property in the file is terminated with a line-terminationcharacter.
- If a line contains only white space, or if the first non-white spacecharacter 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 thefile.
- Each line in the file, unless it is a comment or consists only of whitespace, denotes a single property. A backslash (\) is treatedas the line-continuation character.
- The syntax for a property line consists of a key, a separator, and anelement. Valid separators include the equal sign (=), colon(`:`), and white space ( ).
- The key consists of all characters on the line from the first non-whitespace character to the first separator. Separator characters can beincluded in the key by escaping them with a backslash (\), butdoing this is not recommended, because escaping characters is error prone andconfusing. It is instead recommended that you use a valid separatorcharacter that does not appear in any keys in the properties file.
- White space after the key and separator is ignored until the firstnon-white space character is encountered. All characters remainingbefore the line-termination character define the element.

See the Java documentation for the java.util.Propertiesclass for a full description of the syntax and construction of propertiesfiles.

The following example shows a properties file namedDefaultMessages.properties.

**Figure 1. Sample resource bundle**

```
# Contents of DefaultMessages.properties fileMSG_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 substitution parameters: parm1={0},
parm2={1}, \parm3={2}
```

This file can then be translated into localized versions of the file (forexample, DefaultMessages_de.properties for German andDefaultMessages_ja.properties for Japanese). When the translatedresource bundles are available, they are written to a system-managedpersistent storage medium. Resource bundles are then used to convertthe messages into the requested national language and locale. When amessage logger is obtained from the JRas manager, it can be configured with adefault resource bundle. At run time, the user's locale is used todetermine the properties file from which to extract the message specified by amessage key, thus ensuring that the message is delivered in the correctlanguage. If a default resource bundle is not specified, the msg methodof the RASIMessageLogger interface can be used to specify a resource bundlename.

The application locates the resource bundle based on the file'slocation in the directory structure. For instance, if the resourcebundle is located in thebaseDir/subDir1/subDir2/resources directory andbaseDir is in the classpath, the namesubDir1.subDir2.resources.DefaultMessageis passed to the message logger to identify the resource bundle.

### Creating manager and logger instances

This section provides sample code in which message loggers and traceloggers are obtained in the main method of a standalone application. Toobtain a logger, you first obtain a manager by calling the getManager methodon the com.ibm.websphere.ras.Manager class.You then obtain a message logger by calling createRASIMessageLogger on thereturned manager object, or a trace logger by calling createRASITraceLogger onthe returned manager object. Figure 2 demonstrates these methods.

**Figure 2. Example code: Obtaining a manager, a message logger, and a trace logger**

```
// Import the appropriate JRas and WebSphere packagesimport com.ibm.ras.*;import
com.ibm.websphere.ras.*;// Declare the logger attributes and a group name for trace
loggers. The storage// scope used here depends on the application.static
RASITraceLogger trcLogger = null;static RASIMessageLogger msgLogger = null;// Define
some convenience stringsstatic String svOrg = "My organization name";static String
svProd = "My product name";static String svComponent = "My component name";static
String svClassName = "Fully qualified class name";static java.lang.String groupName =
"MyProduct_someGroup";...public static void main(String[] argv){// Get a reference to
the Manager instance and create the loggers.// Because "Manager" is a common term,
fully qualify it to ensure we// get the right one.com.ibm.websphere.ras.Manager mgr =
com.ibm.websphere.ras.Manager.getManager();msgLogger =
mgr.createRASIMessageLogger(svOrg, svProd, svComponent, svClassName);trcLogger =
mgr.createRASITraceLogger(svOrg, svProd, svComponent, svClassName);// Configure the
message logger with the default resource
bundlemsgLogger.setMessageFile("subDir1.subDir2.resources.DefaultMessages");// Add
the trace logger to a groupmgr.addLoggerToGroup(trcLogger, groupName);}
```

### Using loggers

This section discusses the use of JRas loggers in WebSphereapplications. Message and trace parameters discusses the message and trace parameters usedwith JRas objects. The RASILogger interface discusses the RASILogger interface, The RASIMessageLogger interface discusses the RASIMessageLogger interface,and The RASITraceLogger interface discusses the RASITraceLogger interface. Figure 3 shows examples of using thesemethods.

#### Message and trace parameters

The JRas methods accept parameter types of Object, Object[], andException. The following is a list of parameter types and how they arehandled by the WebSphere implementation of JRas.

- *Primitives*--Primitive data types such as int and long arenot recognized as subclasses of the Object class and cannot be directly passedto JRas methods. A primitive value must be transformed to its propertype (for instance, Integer or Long) before being passed as aparameter.
- *Object*--JRas methods accept members of the Objectclass; the toString method is called on the object and the resultingString is returned. The toString method must therefore be implementedon Objects of traced classes.
- *Object[]*--JRas methods accept members of the Object[]class when two or more Object parameters need to be passed to themethod. The toString method is called on each Object in thearray. Nested arrays (that is, arrays with elements that are alsoarrays) are not supported.
- *Throwable*--JRas methods accept members of the Throwableclass, returning the stack trace of the Throwable object.
- *Arrays of primitives*--An array of primitives (for example,byte[] or int[]) is considered to be an Object by Java; however, becauseof potentially inconsistent processing, it is recommended that members of thearray be converted to String and then passed to the method. If suchconversion is not performed, the results are unpredictable.

#### The RASILogger interface

The RASILogger interface is the base interface for both theRASIMessageLogger and RASITraceLogger classes. This section discussestopics that are common to both of these classes, including the isLoggable,getName and setName, and isSynchronous and setSynchronous methods. See Figure 3 for examples of the classes and methods being used incontext.

The RASILogger interface provides the isLoggable method to determinewhether a logger is currently enabled to log a particular event type.The event type to be checked is passed to the method. The definition isas follows:

```
public boolean isLoggable(long type);
```

where type is a valid message or trace type. See Message and trace event types for a discussion of message and trace types.

The getName and setName methods provide access to logger names.Because all loggers are assigned an unchangeable name by the manager when theyare created, the setName method results in a null operation if used.The getName method can be used at any time to retrieve a logger'sname. The definitions of these methods are as follows:

```
public String getName();public void setName (String name);
```

where name is the logger's name.

The isSynchronous and setSynchronous methods enable applications toconfigure loggers to perform synchronous or asynchronous logging, assumingthat the logger can accept the configuration. The configuration is setby the WebSphere run time, so the setSynchronous method is currentlyimplemented as a null operation. The definitions of these methods areas follows:

```
public boolean isSynchronous();public void setSynchronous(boolean flag);
```

where flag is a Boolean value indicating `True` (forsynchronous logging) or `False` (for asynchronous logging).

#### The RASIMessageLogger interface

The RASIMessageLogger interface provides methods that enable localizablemessage logging. These methods include getMessageFile andsetMessageFile, message, msg, and textMessage. When an instance ofRASIMessageLogger is obtained from the manager, you must provide nonnullstrings that specify the logger's organization name, product name, andcomponent information. These strings are unchangeable for the lifetimeof the logger.

The logger interface includes support for an internal mask that identifieswhich categories of messages are to be logged and which categories are to bedisregarded. The mask is set by the WebSphere run time when the loggeris created.

The getMessageFile method enables you to specify a resource bundle that thelogger uses to localize messages. If the name of the resource bundle isnot specified, a default name is assumed. The setMessageFile enablesyou to configure the message logger with a message file that is used by amessage logged by the message interface. There is no default value forthe message file; if this value is not specified, using the messageinterface can have unpredictable results. See Creating resource bundles and message files for information on resource bundles. The definitionsof the methods are as follows:

```
public String getMessageFile();public void setMessageFile(String file);
```

where file is the name of the resource bundle.

The message method provides flexible access to message strings. Thedefinition of the method is as follows:

```
public void message(long type, Object obj, String methodName, String key,Object
parameter);
```

where:

- type is a valid message type. See Message and trace event types for a discussion of trace types.
- obj is a class name to be passed to the logger. You canpass the class name in the form of either a String or an Object.Passing a String is more efficient and must be used in static methods.For convenience, you can also pass the class name in the form of the`this` object; in this case, the logger retrieves the class namefrom the `this` reference by callingthis.getClass().getName().
- methodName is a valid method name.
- key is the message key of the localizable message. Theresource bundle that was specified by the setMessageFile method is used toretrieve the message text.
- parameter represents an Object that is to be substitutedpositionally into the message text. More than one parametercan be passed. See Message and trace parameters for more information.

The msg method also provides access to message strings; unlike themessage method, it enables you to specify the resource bundle from whichmessage text is to be retrieved. The definition of the method is asfollows:

```
public void msg(long type, Object obj, String methodName, String key,String file,
Object parameter);
```

where:

- type is a valid message type. See Message and trace event types for a discussion of message types.
- obj is a class name to be passed to the logger. You canpass the class name in the form of either a String or an Object.Passing a String is more efficient and must be used in static methods.For convenience, you can also pass the class name in the form of the`this` object; in this case, the logger retrieves the class namefrom the `this` reference by callingthis.getClass().getName().
- methodName is a valid method name.
- key is the message key of the localizable message.
- file is the resource bundle to use when retrieving the messagetext.
- parameter represents an Object that is to be substitutedpositionally into the message text. More than one parametercan be passed. See Message and trace parameters for more information.

The textMessage method enables applications to send text messages that arenot accessed from a resource bundle. This method is intended for use indevelopment environments or environments in which localization support is notrequired. This method is not intended to be used in productioncode. The definition of the method is as follows:

```
public void textMessage(long type, Object obj, String methodName,String text, Object
parameter);
```

where:

- type is a valid message type. See Message and trace event types for a discussion of message types.
- obj is a class name to be passed to the logger. You canpass the class name in the form of either a String or an Object.Passing a String is more efficient and must be used in static methods.For convenience, you can also pass the class name in the form of the`this` object; in this case, the logger retrieves the class namefrom the `this` reference by callingthis.getClass().getName().
- methodName is a valid method name.
- text is the message text. No resource bundle is accessedto provide the text, and the text cannot be localized.
- parameter represents an Object that is to be appended to themessage text. More than one parameter can be passed.See Message and trace parameters for more information.

### The RASITraceLogger interface

The RASITraceLogger interface provides methods that enable generic tracingmechanisms. These methods include entry, exit, trace, andexception. When an instance of RASITraceLogger is obtained from themanager, you must provide nonnull strings that specify the logger'sorganization name, product name, and component information. Thesestrings are unchangeable for the lifetime of the logger.

The logger interface includes support for an internal mask that identifieswhich categories of trace events are to be logged and which categories are tobe disregarded. The mask is set by the WebSphere run time when thelogger is created.

The entry method provides access to trace entry events. Thedefinition of the method is as follows:

```
public void entry(long type, Object obj, String methodName, Object parameter);
```

where:

- type is a valid trace type. See Message and trace event types for a discussion of trace types.
- obj is a class name to be passed to the logger. You canpass the class name in the form of either a String or an Object.Passing a String is more efficient and must be used in static methods.For convenience, you can also pass the class name in the form of the`this` object; in this case, the logger retrieves the class namefrom the `this` reference by callingthis.getClass().getName().
- methodName is a valid method name.
- parameter represents a parameter to be added to the tracetext. See Message and trace parameters for more information.

The exit method provides access to trace exit events. The definitionof the method is as follows:

```
public void exit(long type, Object obj, String methodName, Object retValue);
```

where:

- type is a valid trace type. See Message and trace event types for a discussion of trace types.
- obj is a class name to be passed to the logger. You canpass the class name in the form of either a String or an Object.Passing a String is more efficient and must be used in static methods.For convenience, you can also pass the class name in the form of the`this` object; in this case, the logger retrieves the class namefrom the `this` reference by callingthis.getClass().getName().
- methodName is a valid method name.
- retValue is a return value for the event. See Message and trace parameters for more information.

The trace method provides a way to write text strings as traceevents. The definition of the method is as follows:

```
public void trace(long type, Object obj, String methodName, String text,Object
parameter);
```

where:

- type is a valid trace type. See Message and trace event types for a discussion of trace types.
- obj is a class name to be passed to the logger. You canpass the class name in the form of either a String or an Object.Passing a String is more efficient and must be used in static methods.For convenience, you can also pass the class name in the form of the`this` object; in this case, the logger retrieves the class namefrom the `this` reference by callingthis.getClass().getName().
- methodName is a valid method name.
- text is a text string to be written to the trace eventrecord.
- parameter represents a parameter to be added to the tracetext. See Message and trace parameters for more information.

The exception method provides access to exceptions. The definitionof the method is as follows:

```
public void exception(long type, Object obj, String methodName,Exception exc);
```

where:

- type is a valid trace type. See Message and trace event types for a discussion of trace types.
- obj is a class name to be passed to the logger. You canpass the class name in the form of either a String or an Object.Passing a String is more efficient and must be used in static methods.For convenience, you can also pass the class name in the form of the`this` object; in this case, the logger retrieves the class namefrom the `this` reference by callingthis.getClass().getName().
- methodName is a valid method name.

- exc is an exception whose stack trace is to be written to thetrace event record.

Figure 3 shows an example of using a message logger and a tracelogger.

**Figure 3. Example code: Using a message logger and a trace logger**

```
private void methodX(int x, String y, Foo z){// Trace an entry point. Use the guard
to ensure tracing is enabled. Do this// checking before we waste cycles gathering
parameters to be traced.if (trcLogger.isLoggable(RASITraceEvent.TYPE_ENTRY_EXIT)) {//
Because we want to trace three parameters, package them into an Object[]Object[]
parms = {new Integer(x), y, z};trcLogger.entry(RASITraceEvent.TYPE_ENTRY_EXIT, this,
"methodX", parms);} // ...additional logic here... // A debug or verbose trace
pointif (trcLogger.isLoggable(RASITraceEvent.TYPE_MISC_DATA))
{trcLogger.trace(RASITraceEvent.TYPE_MISC_DATA, this, "methodX", "reached here");} //
... // Call methodY on Foo. Assume that Foo is provided by another vendor or user.//
This method throws no Exceptions, so any run-time exceptions such as a//
NullPointerException coming out of it must be logged as errors.// Although it is not
good practice to put stack traces into message,// it is not explicitly prohibited.try
{z.methodY(...);}catch (Throwable t) {msgLogger.message(RASIMessageEvent.TYPE_ERR,
this, "methodX", "MSG_KEY_01", t);}// ... // Another classification of trace event.
An important state change was// 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");} }
```

# Figures

1. Sample resource bundle
2. Example code: Obtaining a manager, a message logger, and a trace logger
3. Example code: Using a message logger and a trace logger

# Tables

1. JRas message types and their WebSphere equivalents
2. Leveled JRas trace types and their WebSphere equivalents
3. Nonleveled JRas trace types and their WebSphere equivalents

**First Edition (March 2001)**

This softcopy version is based on the printed edition of this book.Some formatting amendments have been made to make this information moresuitable for softcopy.

Order publications through your IBM representative or through the IBMbranch office serving your locality.

# Contents

**Figures**

**Tables**

**Using the Performance Monitoring Infrastructure Client Package**

- Introduction
  - PMI organization and implementation
- PMI client interfaces
  - Data organization and hierarchy
  - PMI interfaces
- Using the PMI client interfaces

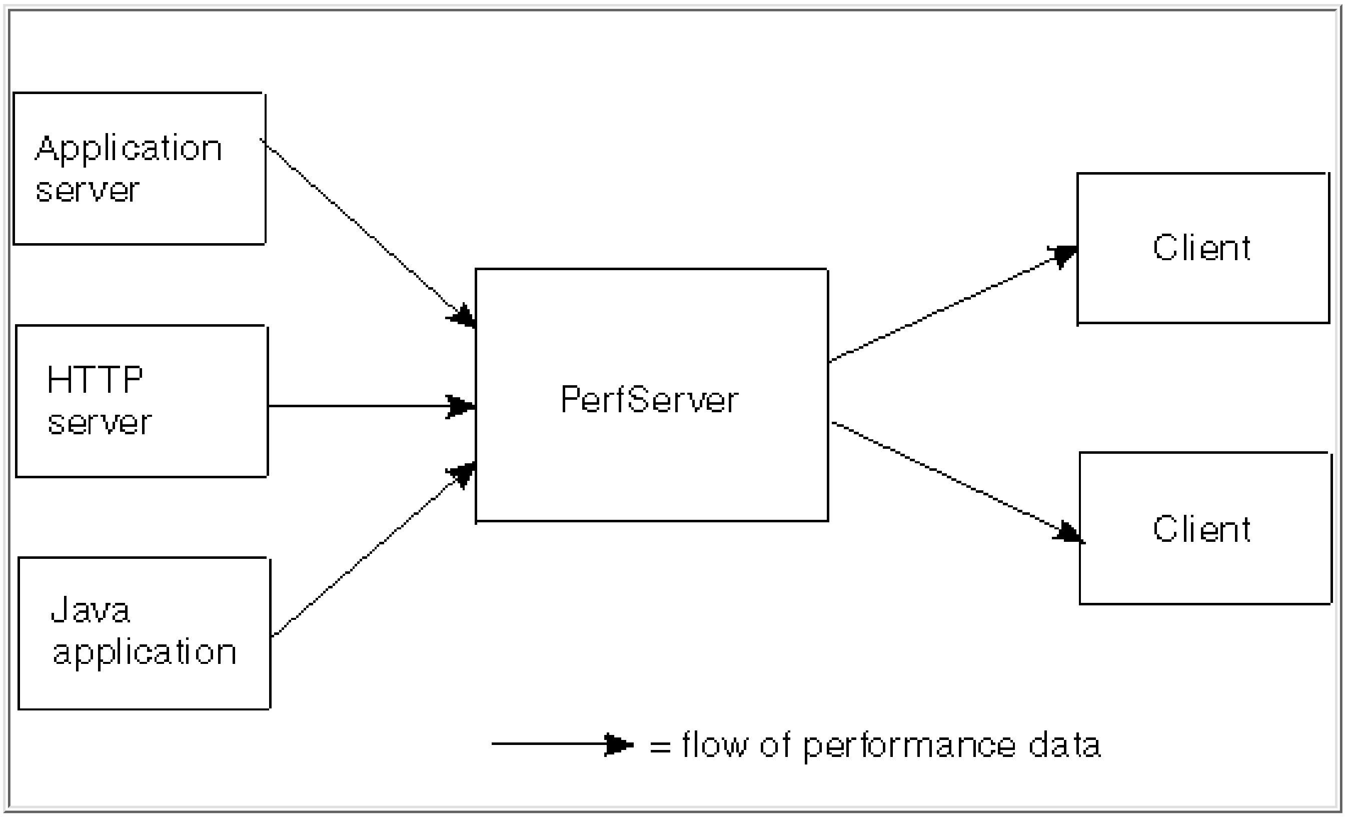
## Introduction

The Performance Monitoring Infrastructure (PMI) is a set of packages andlibraries designed to assist with gathering, delivering, processing, anddisplaying performance data in WebSphere Application Server Advanced Editiondomains. This document discusses the client packages of the PMIapplication programming interface (API) and describes how to use them to writeWebSphere Application Server clients that collect and display performance datafrom servers.

### PMI organization and implementation

PMI follows a client/server architecture. In PMI terms, a server isany application that uses the PMI API to collect performance data;servers can include application servers, HTTP servers, and Javaapplications. WebSphere Application Server provides a server namedPerfServer that is responsible for retrieving performance data from otherservers in the domain and making the data available to interested clients, asshown in Figure 1. A client is an application that receives performancedata from a server or servers and processes the data; clients can includegraphical user interfaces (GUIs) that display performance data in real time,applications that monitor performance data and trigger different eventsaccording to the current values of the data, or any other application thatneeds to receive and process performance data.

**Figure 1. The role of PerfServer in collecting and distributing performance data**



Each piece of performance data has two components, a static component and adynamic component. The static component consists of a name and an ID toidentify the data, as well as other descriptive attributes that assist theclient in processing and displaying the data. The dynamic componentconsists of information that changes over time, such as the current value of acounter and the time stamp associated with that value.

Performance data is classified into the following four types:

- *Numeric data*, consisting of a single numeric value such as aninteger, a long, or a double. It is used to represent data such ascounts and sizes.
- *Statistical data* on a sample space. It consists of thenumber of elements in the sample set, the sum of the elements, and the sum ofsquares. These values can be used to obtain the mean, the variance, andthe standard deviation of the mean. An example of statistical data

isthe response time for each invocation of an enterprise bean.

- *Load data*, which monitors a value as a function of time.Example uses include tracking the number of threads or the number of servicerequests in a queue. Load data tracks the current value, the time thevalue was reached, and the integral over time of the value. Thesevalues can be used to obtain the weighted average for the level over a periodof time. An example of load data is the average size of a databaseconnection pool during a specified time interval.
- *Group data* is a collection of performance data intended to beused by groups. It enables servers to create sets of performance datathat can be retrieved by clients with a single call.

Data is organized by modules; each module has a configuration file inextensible markup language (XML) format that determines itsorganization. The configuration file lists a unique identifier for eachpiece of performance data in the module. A client can use thedata's unique ID to retrieve the data's static information; theserver then sends the dynamic information associated with that data to theclient. A server can track many instances of each type of performancedata--for example, a number of pieces of performance data tracking theaverage response time of bean methods. In this case, each piece ofperformance data shares the same ID, and the server sends additionalidentifying information (for example, the bean's home name) along withthe performance data so that clients can distinguish among the differentinstances.

PMI interfaces with WebSphere administration utilities to enableadministrators to control the amount and level of performance datacollected. You can access the PMI administrative interface by using theAdministrative Console.

## PMI client interfaces

This section discusses PMI's client implementation, including theorganization of data sent to clients and the interfaces clients use toretrieve and process performance data from servers. Performance dataused by PMI's client implementation is referred to as *clientperformance data* (CPD).

### Data organization and hierarchy

PMI data is provided to clients in a hierarchical structure. TheCpdSnapshot object is the root of the hierarchy. Descending from theCpdSnapshot object are node information, server information, moduleinformation, and PerfCollection and CpdData objects. See Figure 2 for a diagram of the data hierarchy. Note that thenode-information and server-information objects contain no performancedata.

**Figure 2. Organization of PMI data**

Organization of PMI data

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

## PMI interfaces

The PMI PerfServer exports the CpdCollection, CpdData, and CpdValueinterfaces to provide performance data to interested clients. The PMIAPI provides the PmiClient interface to enable clients to receive performancedata from servers. For details on these interfaces, see The CpdCollection interface, The CpdData and CpdValue objects, and The PmiClient class. In addition, PMIprovides the CpdEventListener and CpdEvent interfaces to enable clients toregister as listeners, and thus to be informed when new or changed data isavailable at the server; see The CpdEventListener and CpdEvent interfaces for details. Finally, PMI provides the CpdFamilyclass to assist with displaying data in table form; see The CpdFamily class for details.

### The CpdCollection interface

The CpdCollection interface is the base interface to PMI. Itorganizes performance data in the hierarchy described in Data organization and hierarchy. Each member of the hierarchy is an instance ofCpdCollection that contains a number of data members and a number ofCpdCollection children.

The CpdCollection interface extends two other PMI interfaces, CpdXML andCpdEventSender. These interfaces are defined as follows:

**Figure 3. Definitions of the CpdCollection, CpdXML, and CpdEventSender interfaces**

```
public interface CpdCollection extends Serializable, CpdXML,  CpdEventSender {
public PerfDescriptor getPerfDescriptor();     public String getDescription();
public int numDataMembers();     public CpdData[] dataMembers();     public CpdData
getData(int index);     public int numSubcollection();     public CpdCollection[]
subcollections();     public CpdCollection getSubcollection(int i);     public
CpdCollection findCollection(PerfDescriptor pd);     public void
addSubcollection(CpdCollection col);     public CpdCollection getParent();     public
void update(CpdCollection other);     public CpdCollection reset();} public interface
CpdXML {     public String toXML();     public void fromXML(String xmlStr);} public
interface CpdEventSender extends Cloneable {     public void
addCpdEventListener(CpdEventListener al);     public void
removeCpdEventListener(CpdEventListener al);     public void notifyListeners(CpdEvent
evt);     public void notifyListeners(int evt_type);}
```

The update method updates collections of data. To illustrate thefunctionality of this method, assume that the`collection1.update(collection2)` statement is used to updatea data collection named `collection1` with the data in a collectionnamed `collection2`. In this case, the update method works asfollows:

- If `collection1` and `collection2` represent the samecollection (that is, if they are instances of the same PerfDescriptor object,with `collection2` representing a more recent version of thePerfDescriptor object than `collection1`), the update method performsthe following tasks:
    - If any member of `collection2` does not have a correspondingmember in `collection1`, the update method creates a child collectionof `collection1` that contains the member from`collection2`.
    - For each member of `collection2` that has a corresponding memberin `collection1`, the update method updates the member in`collection1` with the corresponding data in`collection2`.

  The update method then returns a value of `true` to thecaller.
- If `collection1` and `collection2` do not represent thesame collection, the update method performs the following tasks:
    - For any member of `collection1` that has a corresponding memberin `collection2`, the update method updates the member in`collection1` with the corresponding data in`collection2`.
    - If `collection2` is a descendant of `collection1`, theupdate method creates a child collection of `collection1` and updateseach member of the child collection with the corresponding data in`collection2`.

  If neither of these conditions is met, the update method returns a valueof `false`.

The PerfDescriptor interface is used to specify the data that the client isinterested in. It includes methods that return node name, server name,module name, collection name, and full name. Its definition is asfollows:

**Figure 4. Definition of the PerfDescriptor interface**

```
public interface PerfDescriptor extends Serializable {     public int getType(); //
Types include node, server, module, instance,          // and data     public String
getNodeName();     public String getServerName();     public String getModuleName();
public String getName(); // Returns node, server, module, instance,          // or
data name, depending on type     public String getFullName(); // Returns a name in
the following form:          // node.server.module.instance.data     public String[]
getPath();     public boolean equals(PerfDescriptor pd);     public boolean
isDescendingFrom(PerfDescriptor pd);     public int[] getDataIds(); // Returns all
data IDs (null, one, or multiple)          // in the descriptor}
```

The PerfDescriptorList class is used to gather data from multiplePerfDescriptor instances. It includes methods to add, remove, and getPerfDescriptor instances. Its definition is as follows:

**Figure 5. Definition of the PerfDescriptorList interface**

```
public class PerfDescriptorList {     public boolean addDescriptor(PerfDescriptor
pd); // If pd is not in the          // list, add it and return true; otherwise,
return false     public boolean removeDescriptor(PerfDescriptor pd); // If pd is in
the          // list, remove it and return true; otherwise, return false     public
int numDescriptors(); // Return the number of PerfDescriptor          // instances in
the list     public PerfDescriptor[] getDescriptors(); // Return all PerfDescriptors
// in an array}
```

### The CpdData and CpdValue objects

The CpdData object is the lowest level in the CPD hierarchy. EachCpdData instance contains all the static information for the performance dataas well as a getValue method to return the data's dynamic information inthe form of an instance of the CpdValue object. The CpdData interfaceprovides an update method to take a reference to a new version of a piece ofdata and update the current object with the new value. The value isupdated only if the new data has the same name as the original object.The CpdData interface also includes an addListener interface to enable dataobjects to register as event listeners; see The CpdEventListener and CpdEvent interfaces for details. The CpdData interface extends the CpdXMLand CpdEventSender interfaces, which are shown in Figure 3.

The definition of CpdData is as follows:

**Figure 6. Definition of the CpdData interface**

```
public interface CpdData extends Serializable, CpdXML, CpdEventSender {     public
PerfDescriptor getDescriptor();     public String getDescription();     public void
setValue(CpdValue value);     public void update(CpdData other);     public CpdValue
getValue();     public Object getParent();     public void setParent(Object parent);
public boolean reset();}
```

A variety of data types extend the CpdValue interface. The interfaceprovides the getValue, getTime, delta, and rate methods to work with datavalues. The definition of CpdValue is as follows:

**Figure 7. Definition of the CpdValue interface**

```
public inteface CpdValue extends Serializable, Cloneable {     public int getType();
public long getTime();     public double getValue();     public CpdValue
delta(CpdValue prev); // return the difference     public CpdValue rate(CpdValue
prev); // return the rate of the difference     public void combine(CpdValue other);
// add another value to this value     public Object clone();}
```

Each client value type extends the CpdValue interface. The specifictypes are listed in Table 1.

**Table 1. CpdValue types and associated methods**

| Type | Method | Description |
|---|---|---|
| CpdInt | int intValue() | Value as an int |
| CpdLong | long longValue() | Value as a long |
| CpdDouble | double doubleValue() | Value as a double |
| CpdStatData | double mean() | Mean of the sample set |
| | int count() | Element count |
| | double sumsquares() | Sum of squares of the elements |
| | double variance() | Variance |
| | double standardDeviation() | Standard deviation |
| | double confidence(int level) | Confidence interval of the mean |
| CpdLoad | double mean() | Time-weighted average value |
| | double getCurrentValue() | Last data point |
| | long getWeight() | Measured time period |

The getValue method retrieves the value and, if possible, converts it to adouble value. If it cannot make the conversion, it returns`Double.NaN`. The values returned by getValue can beused for displaying and graphing data.

The getTime method returns the server time associated with the data.

The delta method takes the current value and a previous value of a piece ofdata, and returns an object that represents the change between thevalues. The delta method also returns a deltaTime value, whichrepresents the time associated with the delta value and the current value ofthe data. The delta method is defined for all objects listed in Table 1. For CpdStatData, the delta between two valuesprovides the statistics on all members of the current sample set, not onmembers of any previous set. The delta method is also defined forgroups. For two groups, `g1` and `g2`, the objectreturned by the statement `g1.delta(g2)` is a group whosemembers include all members common to both `g1` and`g2`. For each member `m1` of group `g1`with a corresponding value of `m2` in `g2`, thecorresponding delta value is represented by`m1.delta(m2)`.

The rate method returns the rate of change. This method is definedfor the CpdInt, CpdLong, and CpdDouble types. If the rate cannot becalculated (for instance, if the method is used with the CpdStatData orCpdLoad types), the original value is returned.

For the CpdLoad object, the mean method returns the time-weighted averageof the value being tracked. It is computed by dividing the integralvalue by the delta time. If the delta time is 0 (zero), the differencebetween the object's current time and its creation time is used.

#### The PmiClient class

The PmiClient class is used by clients to access performance data.It looks up session beans and invokes remote APIs, thus freeing the programmerfrom having to implement these tasks manually. A client can create aninstance of PmiClient and call all subsequent methods on that object.The PmiClient object converts wire-level data to a client-side data collectionhierarchy and exports methods for clients to create PerfDescriptor objects ifthe objects' names are known. If you know the static names for thenode, server, module, instance, or data, you can callpmiClient.createPerfDescriptor to obtain the PerfDescriptor.Otherwise, you can get the names by issuing the listNodes, listServers, andlistMembers methods on PmiClient.

The definition of PmiClient is as follows:

**Figure 8. Definition of the PmiClient class**

```
public class PmiClient {     // Constructor: Look up a PerfRetrieve session bean home
and     //    create a bean object. Do all initialization (for example,     //    get
all configuration files).     //    Default hostName is localhost; default port is
900     //    Default JNDI name for perfRetrieveHome is "PerfRetrieveHome"
PmiClient();     PmiClient(String hostName);     PmiClient(String hostName, String
port);     PmiClient(String hostName, String port, String perfRetrieveHome);      //
The top-level collection of the data hierarchy.     CpdCollection
createRootCollection();      // The following methods serve as wrappers for the
remote     // methods in PerfRetrieve so that users do not need to     // deal with
remote APIs or wire-level data.      // List all nodes in the domain, then call
// PerfDescriptorInstance.getName() to get the node names.     PerfDescriptor[]
listNodes();      // List all servers in a node; pd is the one returned from     //
listNodes. Call PerfDescriptorInstance.getName() to get     // the server names.
PerfDescriptor[] listServers(String nodeName);     PerfDescriptor[]
listServers(PerfDescriptor pd);      // List the members in a server. The returned
PerfDescriptor     // can be passed to the next listMembers call until it     //
returns null (that is, when the leaf node is reached).     PerfDescriptor[]
listMembers(PerfDescriptor pd);      // Get module configuration, which contains all
the static     // information for the data.     PmiModuleConfig[] getConfigs();
PmiModuleConfig[] getConfigs(String nodeName);     PmiModuleConfig getConfig(String
moduleID);      // Retrieve performance data. The following modes are available:
// - Single pd versus an array of pds     // - With or without time interval     // -
Recursive versus nonrecursive (recursive retrieves data     //    for each subgroup
instead of aggregate data)     CpdCollection get(PerfDescriptor pd, boolean
recursive);     CpdCollection get(PerfDescriptor pd, boolean recursive, int time);
CpdCollection[] gets(PerfDescriptorList pds, boolean recursive);     CpdCollection[]
gets(PerfDescriptorList pds, boolean recursive,          int time);
     // Retrieve performance data in XML format     String getXML(PerfDescriptor pd,
boolean recursive);     String getXML(PerfDescriptor pd, boolean recursive, int
time);     String getXML(PerfDescriptorList pds, boolean recursive);     String
getXML(PerfDescriptorList pds, boolean recursive,          int time);      // Convert
data ID and name     public static String getDataName(String moduleID, int dataId);
public static int getDataId(String moduleID, String name);      // Methods to create
a PerfDescriptor, used when you know     // static names     public PerfDescriptor
createPerfDescriptor(){     public PerfDescriptor createPerfDescriptor(String[]
dataPath);     public PerfDescriptor createPerfDescriptor(String[] dataPath,
int dataId);     public PerfDescriptor createPerfDescriptor(String[] dataPath,
int[] dataIds);     public PerfDescriptor createPerfDescriptor(PerfDescriptor parent,
String name);     public PerfDescriptor createPerfDescriptor(PerfDescriptor parent,
int dataId);     public PerfDescriptor createPerfDescriptor(PerfDescriptor parent,
int[] dataIds);     }}
```

### The CpdEventListener and CpdEvent interfaces

The PMI client package provides event and listener interfaces to informclients (for instance, a GUI display) when new or changed data isavailable. The CpdEventObject interface, which extendsjava.util.EventObject, is the parent to the PMI event andlistener interfaces. The CpdEventListener interface, which extendsCpdEventObject, is the interface that objects need to implement to receiveperformance data events. Objects can use the addListener method toregister as event listeners. The definition of the method is asfollows:

```
void addListener(CpdEventListener listener);
```

The definitions of the CpdEventListener and CpdEvent interfaces are asfollows:

**Figure 9. Definitions of the CpdEventListener and CpdEvent interfaces**

```
public interface CpdEventListener {     public void CpdEventPerformed(CpdEvent evt);}
public class CpdEvent {     final static int EVENT_NEW_MEMBER = 0;     final static
int EVENT_NEW_SUBCOLLECTION = 1;     final static int EVENT_NEW_DATA = 2;
private int type;     private Object source = null;      public CpdEvent(Object
source, int type);     public CpdEvent(int type);     public Object getSource();
public int getType();}
```

#### The CpdFamily class

The PMI client provides the CpdFamily class to simplify displaying data ina table. When two data objects have the same module identifier, theyare in the same family and can be displayed in the same table by using thisclass. The definition of CpdFamily is as follows:

**Figure 10. Definition of the CpdFamily class**

```
public class CpdFamily {     static public boolean isSameFamily(CpdData d1, CpdData
d2);     static public boolean isSameRow(CpdData d1, CpdData d2);     static public
boolean isSameColumn(CpdData d1, CpdData d2);     static public boolean
getRow(CpdData d1);     static public boolean getColumn(CpdData d1);     static
public boolean getFamilyName(CpdData d1);}
```

## Using the PMI client interfaces

This section discusses the use of the PMI client interfaces inapplications. The basic programming model is as follows:

1. A client uses the CpdCollection interface to retrieve an initialcollection, or snapshot, of performance data from the server. Thissnapshot, which is called `Snapshot` in this example, is provided ina hierarchical structure as described in Data organization and hierarchy, and contains the current values of all performance datacollected by the server. The snapshot maintains the same structurethroughout the lifetime of the CpdCollection instance.
2. The client processes and displays the data asspecified. Processing and display objects (for example, filters andGUIs) can register as CpdEvent listeners to data of interest; see The CpdEventListener and CpdEvent interfaces for details. When the client receives updated data,all listeners are notified.
3. When the client collects new or changed data (forexample, data collections named `S1`, `S2`, and so on) fromthe server, the client uses the update method to update `Snapshot`with the new data:

   `Snapshot.update(S1);// ...later...Snapshot.update(S2);`
4. Step 2 and Step 3 are repeated through the lifetime of theclient.

Figure 11 lists a sample of PMI client code.

**Figure 11. Example of PMI client code**

```
import com.ibm.websphere.pmi.*;import com.ibm.websphere.pmi.server.*;import
com.ibm.websphere.pmi.client.*; public class PmiTest implements PmiConstants {    //
A test driver   // If arguments are provided:   //   args[0] = node name   //
args[1] = port number   //   args[2] = The JNDI name of PerfRetrieve   //   // Note:
This will not work unless an admin server and    // perfServer are running   //
public static void main(String[] args) {     String hostName = null;     String
portNumber = null;     String homeName = null;     if (args.length >= 1)
hostName = args[0];     if (args.length >=2)        portNumber = args[1];     if
(args.length >=3)        homeName = args[2];      PmiClient pmiClnt = new
PmiClient(hostName, portNumber, homeName);      // Root of PMI data tree
CpdCollection rootCol = pmiClnt.createRootCollection();
     // Set performance descriptor (pd) list     // pdList will include all
PerfDescriptors for data retrieval     PerfDescriptorList pdList = new
PerfDescriptorList();     try {          // If you want to query PmiClient to find
the PerfDescriptor          // you need, you can go through listNodes, listServers,
and          // listMembers to list all the PerfDescriptors and extract          //
the one you want.          PerfDescriptor[] nodePds = pmiClnt.listNodes();
String nodeName = nodePds[0].getName();          System.out.println("after
listNodes:" + nodeName);          PerfDescriptor[] serverPds = pmiClnt.listServers(
nodePds[0].getName());          System.out.println("after listServers");           if
(serverPds == null || serverPds.length == 0) {               System.out.println("NO
app server in node");               return;          }          // For a simple test,
get from the first server          PerfDescriptor[] myPds =
pmiClnt.listMembers(serverPds[0]);          // You can add all pds to
PerfDescriptorList          for (int i = 0; i < myPds.length; i++) {            if
(myPds[i].getModuleName().equals(                "com.ibm.websphere.pmi.beanModule")
|| myPds[i].getModuleName().equals(
"com.ibm.websphere.pmi.connectionPoolModule")            ||
myPds[i].getModuleName.equals(                "com.ibm.websphere.pmi.webAppModule"))
pdList.addDescriptor(myPds[i]);          }            // Or, if you know the data path
you want, you can create your own          String[] thisPath = new
String[]{"thisNode", "thisServer",
"com.ibm.websphere.pmi.transactionModule"};          // Suppose you are interested
only in dataIds 1, 2, and 3          PerfDescriptor thisPd =
pmiClnt.createPerfDescriptor(thisPath,               new int[]{1, 2, 3});
pdList.addDescriptor(thisPd); } catch (Exception ex) {
System.out.println("Exception calling CollectorAE");
ex.printStackTrack();     }
     // Retrieve the data in pdList     CpdCollection[] cpdCols = null;     try {
```

```
for (int i = 0; i < 10; i++) {               java.lang.Thread.sleep(1000);
cpdCols = pmiClnt.gets(pdList, true);               if (cpdCols == null ||
cpdCols.length == 0) {                    System.out.println(
"PMI data return null--possible wrong pds");               }               for (int j
= 0; j < cpdCols.length; j=++) {                    rootCol.update(cpdCols[j]);
report(cpdCols[j]);               }          }     } catch (Exception ex {
System.out.println("Exception to call thread sleep");     }   }    // Simple method
to make sure we are getting the correct 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]);             }        }   }    // Simple method to write
the full name of a pd   private static void printPD(PerfDescriptor pd) {
System.out.println(pd.getFullName());   }}
```

## Figures

1. The role of PerfServer in collecting and distributing performance data
2. Organization of PMI data
3. Definitions of the CpdCollection, CpdXML, and CpdEventSender interfaces
4. Definition of the PerfDescriptor interface
5. Definition of the PerfDescriptorList interface
6. Definition of the CpdData interface
7. Definition of the CpdValue interface
8. Definition of the PmiClient class
9. Definitions of the CpdEventListener and CpdEvent interfaces
10. Definition of the CpdFamily class
11. Example of PMI client code

# Tables

1. CpdValue types and associated methods