WebSphere® Application Server for i5/OS, Version 6.1

Troubleshooting and support

Note

Before using this information, be sure to read the general information under “Notices” on page 407.

Compilation date: May 12, 2006

Contents

How to send your comments

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

- To send comments on articles in the WebSphere Application Server Information Center
  1. Display the article in your Web browser and scroll to the end of the article.
  2. Click on the **Feedback** link at the bottom of the article, and a separate window containing an e-mail form appears.
  3. Fill out the e-mail form as instructed, and click on **Submit feedback** .
- To send comments on PDF books, you can e-mail your comments to: **wasdoc@us.ibm.com** or fax them to 919-254-0206.

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

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Chapter 1. Overview and new features for troubleshooting

Use the links provided in this topic to learn about troubleshooting and problem determination capabilities.

"What is new for troubleshooters" on page 2

This topic provides an overview of new and changed features in troubleshooting tools and support.

Chapter 6, "Diagnosing problems (using diagnosis tools)," on page 89

This topic provides a place to start your search for troubleshooting information.

Troubleshooting overview

Troubleshooting is the process of finding and eliminating the cause of a problem. Whenever you have a problem with your IBM software, the troubleshooting process begins as soon as you ask yourself what happened? A basic troubleshooting strategy at a high level involves:

1. Recording the symptoms.
2. Recreating the problem.
3. Eliminating possible causes.
4. Using diagnostic tools.

Recording the symptoms of the problem

Depending on the type of problem you have, whether it be with your application, your server, or your tools, you might receive a message that indicates something is wrong. Always record the error message that you see. As simple as this sounds, error messages sometimes contain codes that might make more sense as you investigate your problem further. You might also receive multiple error messages that look similar but have subtle differences. By recording the details of each one, you can learn more about where your problem exists.

Recreating the problem

Think back to what steps you were doing that led you to this problem. Try those steps again to see if you can easily recreate this problem. If you have a consistently repeatable test case, you will have an easier time determining what solutions are necessary.

- How did you first notice the problem?
- Did you do anything different that made you notice the problem?
- Is the process that is causing the problem a new procedure, or has it worked successfully before?
- If this worked before what has changed? The change can refer to any type of change made to the system, ranging from adding new hardware or software, to configuration changes to existing software.
- What was the first symptom of the problem you witnessed? Were there other symptoms occurring around that point of time?
- Does the same problem occur elsewhere? Is only one machine experiencing the problem or are multiple machines experiencing the same problem?
- What messages are being generated that could indicate what the problem is?

Eliminating possible causes

Narrow the scope of your problem by eliminating components that are not causing the problem. By using a process of elimination, you can simplify your problem and avoid wasting time in areas that are not culprits. Consult the information in this product and other available resources to help you with your elimination process.

- Has anyone else experienced this problem? Read the information about searching knowledge bases in the information center.
- Is there a fix you can download? Refer to the information center for information about obtaining fixes.
- You can use the WebSphere Application Server Troubleshooting Guide to help you work through the cause of a problem.

Using diagnostic tools

As a more advanced task, there are various tools that you can use to analyze and diagnose problems with your system. To learn how to use these tools see Chapter 6, “Diagnosing problems (using diagnosis tools),” on page 89.

What is new for troubleshooters

This version provides many new features for troubleshooting and servicing the product, with a focus on the ability to automatically detect and recover from problems.

New in Version 6.1! indicates new features or changes implemented at the Version 6.1 level. Unmarked items are Version 6.0 improvements that apply also to Version 6.1, which should interest anyone migrating to Version 6.1 from Version 5.x.

For more detailed information about features that are being replaced or removed in this or future releases, see the *Deprecated and removed features* section in the *Migrating, coexisting, and interoperating* PDF book.

Tools and data capture

Improved ability to manage recovery from failure

To prevent the assignment of new work to an application server that is going through its transaction recovery process, restart the application server in recovery mode.

For more details, see the topic, *Restarting an application server in recovery mode* located in the *Administering applications and their environment* PDF book.

Easier and faster problem determination with diagnostic providers

New in Version 6.1! Diagnostic Providers expose information about running components, enabling administrators to more easily debug problems related to the components. You can detect problems faster and access more information for determining and solving problems.

- Some logged messages contain diagnostic provider IDs (DPIDs), enabling administrators or administrative programs to target requests for information to the components that logged a message.
- Using the administration console, administrators can work with diagnostic providers to dump configuration data, dump state information, or execute any self diagnostic tests that the corresponding component offers.
- Diagnostic provider management APIs are provided to enable programmatic access to diagnostic provider function.

See “Working with Diagnostic Providers” on page 131.

IBM Support Assistant

The IBM Support Assistant (ISA) provides a natural, look-here-first facility for issue resolution. It provides a single point of access to problem determination tools and externally available Support resources for the products you choose. It guides you through exploring and resolving your immediate issue. You can:

- Search multiple IBM and non-IBM locations for the most pertinent support information
- Find and run serviceability tools easily.
- Obtain quick access to appropriate IBM resources such as product pages, support pages and news groups
- Open a service request electronically if you end up needing one.

See “Getting IBM Support Assistant” on page 124.

New troubleshooting technology on the Support site

- MustGather: Read first for all WebSphere Application Server products
- Troubleshooting Guide for WebSphere Application Server

More diagnostic data is captured when failures occur

The first failure data capture (FFDC) feature preserves the information that is generated from a processing failure and returns control to the affected engines. The captured data is saved in a log file for analyzing the problem. FFDC is intended primarily for use by IBM Service.

See “Configuring first failure data capture log file purges” on page 124.

Class Loader Viewer

Class loaders find and load class files. For a deployed application to run properly, the class loaders that affect the application and its modules must be configured so that the application can find the files and resources that it needs. Diagnosing problems with class loaders can be complicated and time-consuming. To help you diagnose and fix problems more quickly, use the administrative console Class Loader Viewer to examine class loaders and the classes loaded by each class loader.

See “Troubleshooting class loaders” on page 89.

The troubleshooting documentation includes extensive support information, including the ability to search live, Web-based support resources by using the customized query fields in the ″Web search″ page.

Messages

Improved message text, new message IDs

Messages for key product components used during installation, migration, and initial configuration have been improved. Additional components have messages now. Message IDs will be changing in a future release. In the meantime, you can configure the application server to use the new message IDs to help prepare any tooling you may have that is reliant on message IDs. The *Troubleshooter reference: Messages* reference section provides a mapping of Version 5.1.x to Version 6.0.x message IDs.

The article "Changing the message IDs used in log files" on page 31 explains how to make your application server use the new message IDs. In addition, the article "Converting log files to use IBM unique Message IDs" on page 32 explains a command to create a log file with new message IDs based on a log file with the older message IDs. This command is provided to ease migration to the new message IDs,

New in Version 6.1! Further work was performed to ensure messages and their component message IDs now correspond to IBM software standards, enabling better results when you use problem determination tools. Message documentation is improved to ensure messages are meaningful and explanatory.

Events

Common Base Events describe system situations

Common Base Events are data structures used to describe situations that occur in the system. Common Base Events are used for various purposes, including representing things such as business events, configuration events, error events, and so on. The WebSphere Application Server now uses Common Base Events as the internal representation of logged messages.

Common Base Events are logged via JSR47 and as such can be received and operated on from JSR47 Handlers. Handlers which are not programmed to the Common Base Event specification will also be able to consume these events as CommonBaseEventLogRecords. Handlers which are programmed to the Common Base Event specification can take advantage of fields within the Common Base Events.

See "The Common Base Event in WebSphere Application Server" on page 65.

Programming to Common Base Event infrastructure

New in Version 6.1! Developers can make use of the Common Base Event infrastructure for logging and to ensure that the events generated are consistent with events generated by the WebSphere runtime, whether or not the developer has chosen to use templates (event factory template or event templates).

See "The Common Base Event in WebSphere Application Server" on page 65.

Memory leak detection

More support for troubleshooting memory leak problems

To help you analyze memory leak problems when memory leak detection occurs, some automated heap dump generation support is available.

Refer to the *Generating and analyzing heap dump* chapter in the *Tuning performance* PDF book.

WebSphere Application Server has implemented a lightweight memory leak detection mechanism that runs within the WebSphere Runtime Performance Advisor framework. This mechanism is designed to provide early detection of memory problems in test and production environments. This framework is not designed to provide analysis of the source of the problem, but rather to provide notification and help generating the information that is required to use analysis tools. The mechanism only detects memory leaks in the Java heap and does not detect native leaks.

For more information on detecting memory leaks, refer to the chapter, *Starting the lightweight memory leak detection* in the *Tuning performance* PDF book.

Memory leak detection enhancements

Recreating memory leaks in a test environment is often difficult. To alleviate the problems associated with reproducing memory leaks in test environments, WebSphere Application Server uses light weight memory leak detection that is designed to run in production systems with minimal performance overhead. This functionality provides early notification of memory leaks, allowing time to diagnose the problem or arrange contingencies before it becomes critical.

New in Version 6.1! Memory leak detection has been improved with new features that provide better support for memory leak diagnosis and troubleshooting.

The Diagnostic Provider Interface provides users with the ability to perform a self diagnostic test that checks for any suspicious memory leaks. This self diagnostic allows problem determination tools to check the status of a memory leak without constantly monitoring WebSphere Application Server for notifications. In addition, new memory leak detection logic has been added to support detection with unbounded heaps. This feature provides more robust early notification for all platforms, even while the Java heap is still expanding.

For more information on detecting memory leaks, refer to the chapter, *Starting the lightweight memory leak detection* in the *Tuning performance* PDF book.

Logging

JRAS is deprecated

The JRAS API is deprecated. Users are directed to use the JSR47 logging infrastructure instead.

For more detailed information about features that are being replaced or removed in this or future releases, see the *Deprecated and removed features* section in the *Migrating, coexisting, and interoperating* PDF book.

Support for Jakarta Commons Logging is added

Jakarta Commons Logging provides a simple logging interface and thin wrappers for several logging systems. WebSphere Application Server version 6.0.2 supports Jakarta Commons Logging by providing a logger, a thin wrapper for the WebSphere Application Server logging facility. The logger can handle both Java Logging (JSR47) and Common Base Event logging objects.

The WebSphere Application Server support for Jakarta Commons Logging does not change interfaces defined by Jakarta Commons Logging.

See “Configuring applications to use Jakarta Commons Logging” on page 36.

Thread names can be included in logs

Thread name has been added to the Advanced log format and Log analyzer trace format. The Log analyzer trace format preserves trace information in the same format as produced by Showlog tool. The advanced log format is available as an output format for the trace log and system out log. The thread name is now included in this format to enable easier correlation with other types of diagnostic data. The log analyzer format is available as an output format for the trace log. The thread name is now included in this format to enable easier correlation with other types of diagnostic data.

See “Interpreting trace output” on page 116.

Java logging framework from JSR47 is exploited

In J2SE 1.4, the Java logging framework was introduced via JSR47. In WebSphere Application Server, messages and trace logged to both JRAS and JSR47 logging APIs are passed into the JSR47 logging infrastructure. This allows JSR47 Handlers connected to the root JSR47 Logger to receive all WebSphere Application Server log content. JSR47 and JRAS Logger levels can be controlled via the admin console troubleshooting section. WebSphere Application Server also builds its logs from the JSR47 framework by connecting its Handlers to the root Logger.

The JSR47 Logging infrastructure allows for flexible pluggability of custom Handlers into the logging infrastructure to enable custom logs. By appropriate configuration, the Handlers can receive WebSphere Application Server’s logged events, and events logged to Loggers instantiated by your applications.

See “Log and trace with Java logging” on page 19.

Troubleshooting by component

Use this page to help resolve a problem within a particular component.

1. Use WebSphere Application Server tracing and logging to help you identify and resolve problems.
2. Determine which component needs troubleshooting.
3. Select the appropriate link from this page and follow the troubleshooting steps.

A client program does not work

What kind of problem are you seeing?

ActiveX client fails to display ASP files, or WebSphere Application Server resources (JSP files, servlet, or HTML pages) or both

A possible cause of this problem is that both IIS for serving Active Server Pages (ASP) files and an HTTP server that supports WebSphere Application Server (such as IBM HTTP Server) are deployed on the same host. This deployment leads to misdirected HTTP traffic if both servers are listening on the same port (such as the default port 80).

To resolve this problem, either:

- Open the IIS administrative panel, and edit the properties of the default Web server to change the port number to a value other than 80
- Install IIS and the HTTP server on separate servers.

For current information available from IBM Support on known problems and their resolution, see the IBM Support page.

IBM Support has documents that can save you time gathering information needed to resolve this problem. Before opening a PMR, see the IBM Support page.

Chapter 2. How do I troubleshoot?

Use this page as a reference of the tutorials included in the WebSphere Application Server, Version 6.1 Information Center. In the information center, hold your cursor over the task icon to see a description of the task. The task preview feature is unavailable for Mozilla Web browsers.

Set traces and logs	ConsoleConsole | For more detailed information on enabling traces by using scripting, see the Troubleshooting with scripting chapter in the *Administering applications and their environment* PDF book.	
Work with message logs	Detailed steps	Show me
Detect hung threads	Detailed steps	
Detect product configuration file problems	Detailed steps	
Troubleshoot problems that occur during a task	Detailed steps	
Debug WebSphere applications during development	Detailed steps	
Add tracing and logging to your applications	Detailed steps	
Collect details for IBM Support	Detailed steps	
Using JSR47 for logging	Detailed steps	

Using JSR47 for logging: Writing a handler	Detailed steps
Using JSR47 for logging: Writing a formatter	Detailed steps
Using JSR47 for logging: Writing a filter	Detailed steps
Using JSR47 for logging: Configuring access logs	Detailed steps
Using Common Base Events for logging	Detailed steps
Creating Common Base Events	Detailed steps

Legend for ″How do I?...″ links

Detailed steps	Show me	Tell me	Guide me	Teach me
Refer to the detailed steps and reference	Watch a brief multimedia demonstration	View the presentation for an overview	Be led through the console pages	Perform the tutorial with sample code
Approximate time: Varies	**Approximate time:** 3 to 5 minutes	**Approximate time:** 10 minutes+	**Approximate time:** 1/2 hour+	**Approximate time:** 1 hour+

Chapter 3. Installing maintenance packages

This topic describes how to use the IBM Update Installer for WebSphere Software to install interim fixes, fix packs, and refresh packs. The Update Installer for WebSphere Software is also known as the Update Installer program, the UpdateInstaller program, and the Update installation wizard.

Use the proper authorizations to successfully install product updates.

When administrative security is enabled on WebSphere Application Server, for example, you must supply the administrative user ID and password before you can update the files.

Use the Update Installer program from a user profile with *ALLOBJ special authority.

Important:

- The user account that originally installed the WebSphere Application Server product to be updated should be used to launch the Installation Wizard to install the Update Installer, and the same user account should be used to launch the Update Installer program to update a product.
  - When a different user account uses the *updi_root* location, that user account must have reading and running access to that location. It must also have writing access to the *updi_root*/logs directory and its subdirectory.
  - When a different user account is used to update the target WebSphere Application Server product location, that user account must have full access (reading, writing, and running) to the target location where a maintenance package is to be applied.
- Make sure that no processes from any users are locking any files in the target location where a maintenance package is to be installed.

The Update Installer wizard is an InstallShield for Multiplatforms wizard that runs with in silent mode with a response file.

The following descriptions contain reference information about installing interim fixes, fix packs, and refresh packs on WebSphere Application Server products and components:

Overview of the installation procedure

1. Download, unpack, and install the Update Installer for WebSphere Software; or install the Update Installer that is on the WebSphere Application Server supplements disc.
2. Download the most current version of the interim fix, fix pack, or refresh pack file from the Support site.
3. Download the interim fix, fix pack, or refresh pack from the Support Web site into the maintenance directory.
4. Use the Update Installer to install the interim fix, fix pack, or refresh pack.

   The Update Installer creates a backup file in the *app_server_root*/properties/version/nif/backup directory.

Updating existing profiles in WebSphere Application Server products

The Update Installer updates the core product files in a WebSphere Application Server product. Service in a maintenance package might update the following files in the installation root directory:

- JAR files in the lib directory
- Scripts in the bin directory
- Profile templates

Viewing the fix level of the product

Use the versionInfo command and the historyInfo command in the bin directory of the installation root directory to display the exact fix and version level of the product. However, do not use either command while installing or uninstalling a maintenance package.

Important: See the release notes for your product for the latest information about the Update Installer.

The following procedure describes how to install a maintenance package.

1. Log on as a user profile with *ALLOBJ special authority.
2. Install the product that you intend to update.

   You have very likely already installed the software that you are now updating. But if not, install the software now.
3. **Optional:** Install a new version of the Update Installer.

   Back up and uninstall any older copy of the Update Installer before downloading and installing the current Update Installer. To use a newer version of the Update Installer, you must first remove the older version.

   a. Back up any files and subdirectories in the *updi_root*/maintenance directory if necessary.
   b. Uninstall the older version of the Update Installer using the program under *updi_root*/uninstall.
   c. Download, unpack, and install the Update Installer for WebSphere Software; or install the Update Installer that is on the WebSphere Application Server supplements disc.
4. Download the maintenance package *.pak file from the Support Web site into the maintenance directory.

   Tip: Do not attempt to unzip or unpack the *.pak file.
5. Stop all processes that use the WebSphere Application Server product.

   Before installing or uninstalling interim fixes, fix packs, and refresh packs on a machine, stop all Java processes on the machine that use the WebSphere Application Server product.

   WebSphere Application Server processes include application server processes, such as the process created when server1 is running.
6. Verify that the following prerequisite conditions are met:
   - All of the product hardware and software prerequisites exist.

     The official statement of supported hardware and software is on the Supported hardware and software Web site.
   - The WebSphere software that you are updating is correctly installed and is not corrupt.
   - The user has *ALLOBJ special authority.
7. Use the Update Installer to install the maintenance package.

   Issue the following command to use the silent interface:

Table 1. Update installer command for installing in silent mode

Command example	Type of installation	Description
update -options "responsefiles/*file_name*"	Silent mode with an options file	Overrides all default values with values that you specified in the options response file. Always use a response file that is based on the response file under *updi_root*/responsefiles.

This procedure results in installing maintenance packages to update WebSphere software.

After installing all maintenance packages, continue to use your WebSphere software.

install.txt

The Update Installer for WebSphere Software can use an options response file to install maintenance packages from a command line interface.

The install.txt file has one directive that identifies the backup file for installing a service update. Comments in the file describe how to set the string value.

The Update Installer for WebSphere Software wizard reads the options file to determine installation choices. The Update Installer installs the maintenance package in silent mode instead of displaying a graphical user interface.

Location of the response file

The sample options response file is named install.txt. The file is in the *updi_root*/responsefiles directory after you install the Update Installer for WebSphere Software into the installation root directory of the WebSphere software product.

Installing silently

The options file supplies the values to the Update installer wizard when installing silently.

The following command uses a copy of the options file named myresponsefile.txt to provide installation option responses during a silent installation:

```
update -options responsefiles/myresponsefile.txt
```

There is no need to specify the -silent parameter as it is already defined in the update script.

Response file user entry validation

In a silent installation, response file validation is coded into the installation. If the validation does not pass, the failure is recorded in the log files in the *app_server_root*/logs/update/tmp directory.

Location of the maintenance package to be installed

Default directive setting

-W maintenance.package=""

Valid setting

You must set this directive to the location of the maintenance package PAK file. For example, you might specify the following location:

```
/QIBM/ProdData/WebSphere/updateinstaller/V61/UPDI/maintenance/PQ20029.pak
```

Error identifiers:

- Maintenance package *maintenance_package_name* is already installed on the system.
- Selected product is not supported.
- Configuration failed. The config action that failed was: *configuration_action*.
- Install the following prerequisite APARs before installing the current maintenance to the target product: *list_ of_ prerequisite_ maintenance_ packages_ to_ install*
- Install the following prerequisite maintenance packages before installing the package you are currently attempting to install: *list_ of_ prerequisite_ maintenance_ packages_ to_ install*
- Uninstall the following APARs before applying the current maintenance to the target product: *list_ of_ prerequisite_ maintenance_ packages_ to_ uninstall*
- Uninstall the following maintenance packages before applying the current maintenance to the target product: *list_ of_ prerequisite_ maintenance_ packages_ to_ uninstall*
- Unable to locate the correct version of *the_update_installer*. Looking for version *version_identifier*.
- *Maintenance_package* is not a valid maintenance package.

Product location

Default directive setting

-W product.location="*SPECIFY_PRODUCT_INSTALL_LOCATION_HERE*"

Valid setting

Set this directive to the installation root directory of the product. For example, you might specify the following location:

```
/QIBM/ProdData/WebSphere/AppServer/V61/Express
```

Error identifiers:

- Maintenance package *maintenance_package_name* is already installed on the system.
- Selected product is not supported.
- Configuration failed. The config action that failed was: *configuration_action*.
- Install the following prerequisite APARs before installing the current maintenance to the target product: *list_ of_ prerequisite_ maintenance_ packages_ to_ install*
- Install the following prerequisite maintenance packages before installing the package you are currently attempting to install: *list_ of_ prerequisite_ maintenance_ packages_ to_ install*
- Uninstall the following APARs before applying the current maintenance to the target product: *list_ of_ prerequisite_ maintenance_ packages_ to_ uninstall*
- Uninstall the following maintenance packages before applying the current maintenance to the target product: *list_ of_ prerequisite_ maintenance_ packages_ to_ uninstall*
- Unable to locate the correct version of *the_update_installer*. Looking for version *version_identifier*.
- *Maintenance_package* is not a valid maintenance package.
- *Alternate_product_directory* could not be validated as an existing directory.

Usage notes

- The file is not a read-only file.
- Edit this file directly with your flat file editor of choice, such as Kate on SLES or WordPad on a Windows platform.
- The file must exist to perform a silent installation. The Update installer wizard reads this file to determine installation parameters. Provide the fully qualified file path to the backup file.
- Save the copy of the options file in the responsefiles directory for best results.

Example install.txt file

Edit the version of the file that is included in the Update Installer for WebSphere Software ZIP file. The following example is not guaranteed to be an accurate representation of the actual file.

```
################################################################################
#
# This is the silent install response file for installing maintenance packages
# using the update installer.
#
# A common use of an options file is to run the wizard in silent mode. This lets
# the options file author specify wizard settings without having to run the
# wizard in graphical or console mode. To use this options file for silent mode
# execution, *uncomment* and modify the parameters defined within.
#
# Use the following command line when running the wizard from the update
# installer directory:
#
#    update -options responsefiles/install.txt
#
# Please enclose all values within a single pair of double quotes.
#
################################################################################


################################################################################
```

```
#
# Used to input the maintenance package full filename specification to be installed.
# Edit as appropriate.
#
# ie. -W maintenance.package="/QIBM/ProdData/WebSphere/UpdateInstaller/V61/UPDI/maintenance/PQ20029.pak"
#
# Note: If no package is specified, a default of the last downloaded maintenance
# package will be used (based on timestamp).
#
#-W maintenance.package=""


################################################################################
#
# Used to input the product install location that will be updated.
#
# ie. -W product.location="/QIBM/ProdData/WebSphere/AppServer/V61/Express"
#
# Note: The product install location should always been specified, and it should
# always be the full path.
#
-W product.location=""



################################################################################
#
# Do not edit these values.
#
-W update.type="install"
```

Updating the Update Installer for WebSphere Software

To get updates for the Update Installer for WebSphere Software, download the most current version from the support site and reinstall it.

Important: Only one copy of the Update Installer should be installed on your system at any one time for use with all Version 6.x products. Before installing a newer version of the Update Installer, you must first remove the existing Update Installer.

The Update Installer ships on the Version 6.1 supplements disk. The Update Installer undergoes regular maintenance and offers updated versions on the WebSphere software support pages.

See Recommended Updates for WebSphere Application Server. Look under your release of the WebSphere software for a link to the Update Installer download page.

This topic describes how to update the Update Installer after you installed it into a product.

1. Uninstall the Update Installer.

   Run the **uninstall** command in the *app_server_root*/bin directory.
2. Download the updated Update Installer files from the product Support site.

   For example, see Recommended Updates for WebSphere Application Server to locate the Update Installer download page.
3. Unpack the downloaded file to reveal the install program.
4. Install the new Update Installer.
   - For remote GUI installation, do the following:
     a. Open Windows Explorer and select your disc drive.
     b. Click the *downloaded_path*\UpdateInstaller\install.exe file to start the InstallShield for Multiplatforms (ISMP) program.
   - For local silent QSH installation, run the **INSTALL** command from Qshell.
     a. On a CL command line, run the **STRQSH** command to start the Qshell command shell.

b. Issue the **INSTALL** command to start the installation program.

```
cd downloaded_path/UpdateInstaller
INSTALL -options path/responsefile
```

The Update Installer is installed under the root directory that you specified during installation.

See Chapter 3, “Installing maintenance packages,” on page 11 for information about using the Update Installer to install maintenance packages.

Chapter 4. Debugging applications

To debug your application, you must use a development environment like Application Server Toolkit or Rational Application Developer to create a Java project. You must then import the program that you want to debug into the project. By following the steps below, you can import the WebSphere Application Server examples into a Java project.

Two debugging styles are available:

- **Step-by-step** debugging mode prompts you whenever the server calls a method on a Web object. A dialog lets you step into the method or skip it. In the dialog, you can turn off step-by-step mode when you are finished using it.
- **Breakpoints** debugging mode lets you debug specific parts of programs. Add breakpoints to the part of the code that you must debug and run the program until one of the breakpoints is encountered.

Breakpoints actually work with both styles of debugging. Step-by-step mode just lets you see which Web objects are being called without having to set up breakpoints ahead of time.

You do not need to import an entire program into your project. However, if you do not import all of your program into the project, some of the source might not compile. You can still debug the project. Most features of the debugger work, including breakpoints, stepping, and viewing and modifying variables. You must import any source that you want to set breakpoints in.

The inspect and display features in the source view do not work if the source has build errors. These features let you select an expression in the source view and evaluate it.

1. Create a Java Project by opening the New Project dialog.
2. Select **Java** from the left side of the dialog and **Java Project** in the right side of the dialog.
3. Click **Next** and specify a name for the project, for example, WASExamples.
4. Click **Finish** to create the project.
5. Select the new project, choose **File > Import > File System**, then **Next** to open the import file system dialog.
6. Browse the directory for files.

   Go to the following directory: *profile_root*/installedApps/*node_name*/DefaultApplication.ear/DefaultWebApplication.war.
7. Select DefaultWebApplication.war in the left side of the Import dialog and then click **Finish**. This imports the JavaServer Pages files and Java source for the examples into your project.
8. Add any JAR files needed to build to the Java Build Path.

   Select **Properties** from the right-click menu. Choose the Java Build Path node and then select the Libraries tab. Click **Add External JARs** to add the following JAR files:
   - *profile_root*/installedApps/*node_name*/DefaultApplication.ear/Increment.jar.

     When you have added this JAR file, select it and use the **Attach Source** function to attach the Increment.jar file because it contains both the source and class files.
   - *app_server_root*/lib/j2ee.jar
   - *app_server_root*/lib/pagelist.jar
   - *app_server_root*/lib/webcontainer.jar

   Click **OK** when you have added all of the JARs.
9. You can set some breakpoints in the source at this time if you like, however, it is not necessary as step-by-step mode will prompt you whenever the server calls a method on a Web object. Step-by-step mode is explained in more detail below.
10. To start debugging, you need to start the WebSphere Application Server in debug mode and make note of the JVM debug port. The default value of the JVM debug port is 7777.

11. When the server is started, switch to the debug perspective by selecting **Window > Open Perspective > Debug**. You can also enable the debug launch in the Java Perspective by choosing **Window > Customize Perspective** and selecting the **Debug** and **Launch** checkboxes in the **Other** category.
12. Select the workbench toolbar **Debug** pushbutton and then select **WebSphere Application Server Debug** from the list of launch configurations. Click the **New** pushbutton to create a new configuration.
13. Give your configuration a name and select the project to debug (your new WASExamples project). Change the port number if you did not start the server on the default port (7777).
14. Click **Debug** to start debugging.
15. Load one of the examples in your browser. For example: `http://your.server.name:9080/hitcount`

To learn more about debugging, launch the Application Server Toolkit, select **Help > Help Contents** and choose the **Debugger Guide bookshelf** entry. To learn about known limitations and problems that are associated with the Application Server Toolkit, see the Application Server Toolkit release notes. For current information available from IBM Support on known problems and their resolution, see the IBM Support page.

IBM Support has documents that can save you time gathering information needed to resolve this problem. Before opening a PMR, see the Must gather documents page for information to gather to send to IBM Support.

Debugging components in the Application Server Toolkit

The Application Server Toolkit, included with the WebSphere Application Server on a separately-installable CD, includes debugging functionality that is built on the Eclipse workbench. Documentation for the Application Server Toolkit is provided with that product. To learn more about the debug components, launch the Application Server Toolkit, select **Help > Help Contents** and choose the **Debugger Guide bookshelf** entry.

The Application Server Toolkit includes the following:

The WebSphere Application Server debug adapter
: which allows you to debug Web objects that are running on WebSphere Application Server and that you have launched in a browser. These objects include enterprise beans, JavaServer Pages files, and servlets.

The JavaScript debug adapter
: which enables server-side JavaScript debugging.

The Compiled language debugger
: which allows you to detect and diagnose errors in compiled-language applications.

The Java development tools (JDT) debugger
: which allows you to debug Java code.

All of the debug components in the Application Server Toolkit can be used for debugging locally and for remote debugging. To learn more about the debug components, launch the Application Server Toolkit, select **Help > Help Contents** and choose the **Debugger Guide bookshelf** entry.

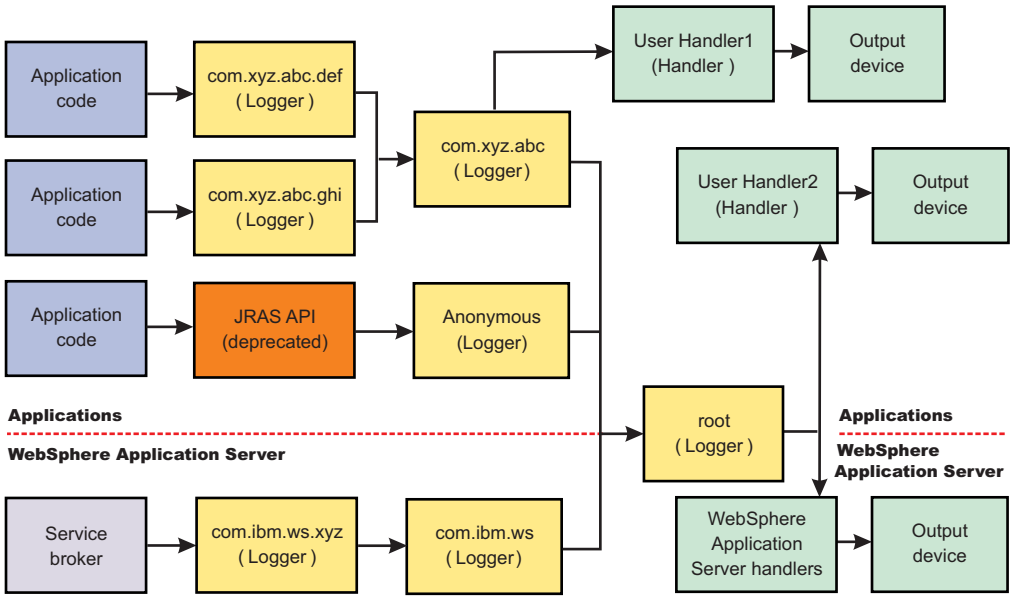
Chapter 5. Add logging and tracing to your application

You can add logging and tracing to applications to help analyze performance and diagnose problems in WebSphere Application Server.

Deprecation: The JRas framework that is described in this information center is deprecated. However, you can achieve the same results using Java logging.

Designers and developers of applications that run with or under WebSphere Application Server, such as servlets, JavaServer Pages (JSP) files , enterprise beans, client applications, and their supporting classes, might find it useful to use Java logging for generating their application logging.

This approach has advantages over adding `System.out.println` statements to your code:
- Your messages are displayed in the WebSphere Application Server standard log files, using a standard message format with additional data, such as a date and time stamp that are added automatically.
- You can more easily correlate problems and events in your own application to problems and events that are associated with WebSphere Application Server components.
- You can take advantage of the WebSphere Application Server log file management features.



1. To use Java logging, configure properties using the administrative console.
2. Customize the properties to meet your logging needs. For example, enable or disable a particular log, specify the number of logs to be kept, and specify a format for log output.
3. Restart the application server after making static configuration changes.

Log and trace with Java logging

Java logging is the logging toolkit that is provided by the java.util.logging package. Java logging provides a standard logging API for your applications.

The application server redirects the system streams at the server startup. There is no way to allow the application to output logging to the console because the system streams can not be obtained by the application. If you would like to use console to monitor the application without using the console handler, you can either monitor the `SystemOut.log` file, or monitor a file created by another file handler.

Note: The application server uses Java logging internally and therefore certain restrictions apply for using system streams with this logging API by applications. During server startup, the standard output and error streams are replaced with special streams that write to the logging infrastructure, in order to include the output of the system streams in the log files. Because of this, applications can not use`java.util.logging.ConsoleHandler`, or any handler writing to `System.err` or `System.out` streams, attached to the root logger. If the user does attache the handler to the root logger, an infinite loop is created within the logging infrastructure, leading to stack overflow and server crash.

If the use of a handler that writes to system streams is necessary, attach it to a non-root logger so that it does not publish log records to parent handlers. The data written to the system streams is then formatted and written to the corresponding system stream log file. To monitor what is being written system streams, the configured log files (`SystemOut.log` and `SystemErr.log` by default) can be monitored.

Developing, deploying and maintaining applications are complex tasks. When an application encounters an unexpected condition, it might not be able to complete a requested operation. You might want the application to inform the administrator that the operation failed and tell the administrator why the operation failed. This information enables the administrator to take the proper corrective action. Application developers might need to gather detailed information that relates to the path of a running application to determine the root cause of a failure that is due to a code bug. The facilities that are used for these purposes are typically referred to as *logging* and *tracing*.

Message logging (messages) and diagnostic trace (trace) are conceptually similar, but do have important differences. These differences are important for application developers to understand to use these tools properly. The following operational definitions of messages and trace are provided.

Message
A message entry is an informational record that is intended for end users, systems administrators, and support personnel to view. The text of the message must be clear, concise, and interpretable by an end user. Messages are typically localized and displayed in the national language of the end user. Although the destination and lifetime of messages might be configurable, enable some level of message logging in normal system operation. Use message logging judiciously because of performance considerations and the size of the message repository.

Trace A trace entry is an information record that is intended for service engineers or developers to use. As such, a trace record might be considerably more complex, verbose, and detailed than a message entry. Localization support is typically not used for trace entries. Trace entries can be fairly inscrutable, understandable only by the appropriate developer or service personnel. It is assumed that trace entries are not written during normal runtime operation, but can be enabled as needed to gather diagnostic information.

- To use Java logging, see “Configuring logging properties using the administrative console” on page 61
- See the Java documentation for the java.util.logging class for a full description of the syntax and the construction of logging methods.

Loggers

Loggers are used by applications and runtime components to capture message and trace events.

When situations occur that are significant either due to a change in state, for example when a server completes startup or because a potential problem is detected, such as a timeout waiting for a resource, a message is written to the logs. Trace events are logged in debugging scenarios, where a developer needs a clear view of what is occurring in each component to understand what might be going wrong. Logged events are often the only events available when a problem is first detected, and are used during both problem recovery and problem resolution.

Loggers are organized hierarchically. Each logger can have zero or more child loggers.

Loggers can be associated with a resource bundle. If specified, the resource bundle is used by the logger to localize messages that are logged to the logger. If the resource bundle is not specified, a logger uses the same resource bundle as its parent.

You can configure loggers with a level. If specified, the level is compared by the logger to incoming events. The events that are less severe than the level set for the logger are ignored by the logger. If the level is not specified, a logger takes on the level that is used by its parent. The default level for loggers is Level.INFO.

Loggers can have zero or more attached handlers. If supplied, all events that are logged to the logger are passed to the attached handlers. Handlers write events to output destinations such as log files or network sockets. When a logger finishes passing a logged event to all of the handlers that are attached to that logger, the logger passes the event to the handlers that are attached to the parents of the logger. This process stops if a parent logger is configured not to use its parent handlers. Handlers in WebSphere Application Server are attached to the root logger. Set the useParentHandlers logger property to `false` to prevent the logger from writing events to handlers that are higher in the hierarchy.

Loggers can have a filter. If supplied, the filter is invoked for each incoming event to tell the logger whether or not to ignore it.

Applications interact directly with loggers to log events. To obtain or create a logger, a call is made to the Logger.getLogger method with a name for the logger. Typically, the logger name is either the package qualified class name or the name of the package that the logger is used by. The hierarchical logger namespace is automatically created by using the dots in the logger name. For example, the com.ibm.websphere.ras logger has a com.ibm.websphere parent logger, which has a com.ibm parent. The parent at the top of the hierarchy is referred to as the *root logger*. This root logger is created during initialization. The root logger is the parent of the com logger.

Loggers are structured in a hierarchy. Every logger, except the root logger, has one parent. Each logger can also have 0 or more children. A logger inherits log handlers, resource bundle names, and event filtering settings from its parent in the hierarchy. The logger hierarchy is managed by the LogManager function.

Loggers create log records. A log record is the container object for the data of an event. This object is used by filters, handlers, and formatters in the logging infrastructure.

The logger provides several sets of methods for generating log messages. Some log methods take only a level and enough information to construct a message. Other, more complex logp (log precise) methods support the caller in passing class name and method name attributes, in addition to the level and message information. The logrb (log with resource bundle) methods add the capability of specifying a resource bundle as well as the level, message information, class name, and method name. Using methods such as severe, warning, fine, finer, and finest you can log a message at a particular level. For more information on logging and how to use it in your applications read “Using loggers in an application” on page 25. For a complete list of methods, see the java.util.logging documentation at http://java.sun.com/j2se/.

Log handlers

Log handlers write log record objects to output devices like log files, sockets, and notification mechanisms.

Loggers can have zero or more attached handlers. All objects that are logged to the logger are passed to the attached handlers, if handlers are supplied.

You can configure handlers with a level. The handler compares the level that is specified in the logged object to the level that is specified for the handler. If the level of the logged object is less severe than the level set in the handler, the object is ignored by the handler. The default level for handlers is ALL.

Handlers can have a filter. If a filter is supplied, the filter is invoked for each incoming object to tell the handler whether or not to ignore it.

Handlers can have a formatter. If a formatter is supplied, the formatter controls how the logged objects are formatted. For example, the formatter can decide to first include the time stamp, followed by a string representation of the level, followed by the message that is included in the logged object. The handler writes this formatted representation to the output device. Read “java.util.logging custom formatters” on page 34 for information on using a custom formatter in your applications.

Both loggers and handlers can have levels and filters, and a logged object must pass all of these elements to be output. For example, you can set the logger level to FINE, but if the handler level is set at WARNING, only WARNING level messages are displayed in the output for that handler. Conversely, if your log handler is set to output all messages (level=All), but the logger level is set to WARNING, the logger never sends messages lower than WARNING to the log handler.

Log levels

Levels control which events are processed by Java logging. WebSphere Application Server controls the levels of all loggers in the system.

The level value is set from configuration data when the logger is created and can be changed at run time from the administrative console. If a level is not set in the configuration data, a level is obtained by proceeding up the hierarchy until a parent with a level value is found. You can also set a level for each handler to indicate which events are published to an output device. When you change the level for a logger in the administrative console, the change is propagated to the children of the logger.

Levels are cumulative; a logger can process logged objects at the level that is set for the logger, and at all levels above the set level. Valid levels are:

Level	**Content / Significance**
Off	No events are logged.
Fatal	Task cannot continue and component cannot function.
Severe	Task cannot continue, but component can still function
Warning	Potential error or impending error
Audit	Significant event affecting server state or resources
Info	General information outlining overall task progress
Config	Configuration change or status
Detail	General information detailing subtask progress
Fine	Trace information - General trace + method entry / exit / return values
Finer	Trace information - Detailed trace
Finest	Trace information - A more detailed trace - Includes all the detail that is needed to debug problems
All	All events are logged. If you create custom levels, All includes your custom levels, and can provide a more detailed trace than Finest.

For instructions on how to set logging levels, see “Configuring logging properties using the administrative console” on page 61

Note: Trace information, which includes events at the Fine, Finer and Finest levels, can be written only to the trace log. Therefore, if you do not enable diagnostic trace, setting the log detail level to Fine, Finer, or Finest does not effect the logged data.

Log filters

Log filters help control more detailed logging settings that are not handled by usual log level settings.

A filter provides an optional, secondary control over what is logged, beyond the control that is provided by setting the level. Applications can apply a filter mechanism to control logging output through the logging APIs. An example of filter usage is to suppress all the events with a particular message key.

A filter is attached to a logger or log handler using the appropriate setFilter method. Read “java.util.logging custom filters” on page 34 for information on implementing custom filters. For a complete list of filter methods, see the java.util.logging documentation at http://java.sun.com/j2se/

Log formatters

Log formatters format log messages so they can be used by various log handlers.

Handlers can be configured with a log formatter that knows how to format log records. The event, which is represented by the log record object, is passed to the appropriate formatter by the handler. The formatter returns formatted output to the handler, which writes the output to the output device.

The formatter is responsible for rendering the event for output. This formatter uses the resource bundle that is specified in the event to look up the message in the appropriate language.

Formatters are attached to handlers using the setFormatter method.

You can find the java.util.logging documentation at http://java.sun.com/j2se/.

Logging properties for an application

Use the Logger.properties file to set logger attributes for specific loggers.

The properties file is loaded the first time that the Logger.getLogger(*logger_name*) method is called within an application.

Important: The name of the Logger.properties file is case sensitive. Use a capital ″L″ in the file name.

When an application calls the Logger.getLogger method for the first time, all the available logger properties files are loaded. Applications can provide Logger.properties files in:

- the META-INF directory of the Java archive (JAR) file for the application
- directories included in the class path of an application module
- directories included in the application class path

The properties file contains two categories of parameters, logger control and logger data:

- Logger control information
  - Minimum localization level: The minimum LogRecord level for which localization is attempted
  - Group: The logical group that this component belongs to
  - Event factory: The Common Base Event template file to use with the event factory. The naming convention for this template is the fully qualified component name, with a file extension of `.event.xml`. For example, a template that applies to the com.ibm.compXYZ package is called `com.ibm.compXYZ.event.xml`.
- Logger data information

– Product name
– Organization name
– Component name
– Extensions and additional properties

Syntax of the Logger.properties file

Use the following syntax to set logger properties:

```
<logger base name>.<property>=value
```

where:

logger base name is the starting part of the logger name to which the property applies. All loggers with names starting with this string have the property applied.

property is one of the following properties:

- organization
- product
- component
- minimum_localization_level
- group
- eventfactory

Sample Logger.properties file

In the following sample, the com.ibm.xyz.MyEventFactory event factory is used by any loggers in the com.ibm.websphere.abc package or any sub packages that do not override this value in their configuration file.

```
com.ibm.websphere.abc.eventfactory=com.ibm.xyz.MyEventFactory
```

Group Logger.properties file

In the following example, the group is MyTraceGroup and the components are com.ibm.stuff and com.ibm.morestuff:

```
com.ibm.stuff.group=MyTraceGroup
com.ibm.morestuff.group=MyTraceGroup
```

Sample security policy for logging

Set up a security policy to allow your applications to modify logging and handler properties.

The sample security policy that follows grants access to the file system and runtime classes. Include this security policy, with the entry `permission java.util.logging.LoggingPermission "control"`, in the META-INF directory of your application if you want your applications to programmatically alter controlled properties of loggers and handlers. The META-INF file is located in the following locations for the different module types:

EJB projects	ejbModule/META-INF/MANIFEST.MF
Application client projects	appClientModule/META-INF/MANIFEST.MF
Dynamic Web projects	WebContent/META-INF/MANIFEST.MF
Connector projects	connectorModule/META-INF/MANIFEST.MF

Below is a sample security policy that grants permission to modify logging properties:

```
////////////////////////////////////////////////
//
// WebSphere Application Server Security Policy
//
////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////
// Allow all access to the file system and runtime classes
////////////////////////////////////////////////////////////////////////
grant codeBase "file:${application}" {
   permission java.util.logging.LoggingPermission "control";
};
```

Using loggers in an application

This topic describes how to use Java logging within an application.

To create an application using Java logging, perform the following steps:

1. Create the necessary handler, formatter, and filter classes if you need your own log files.
2. If localized messages are used by the application, create a resource bundle, as described in “Creating log resource bundles and message files” on page 29.
3. In the application code, get a reference to a logger instance, as described in “Using a logger.”
4. Insert the appropriate message and trace logging statements in the application, as described in “Using a logger.”

Using a logger

You can use Java logging to log messages and add tracing.

Use WsLevel.DETAIL level and above for messages, and lower levels for trace. The WebSphere Application Server Extension API (the com.ibm.websphere.logging package) contains the WsLevel class.

For messages use:

```
WsLevel.FATAL
Level.SEVERE
Level.WARNING
WsLevel.AUDIT
Level.INFO
Level.CONFIG
WsLevel.DETAIL
```

For trace use:

```
Level.FINE
Level.FINER
Level.FINEST
```

1. Use the logp method instead of the log or the logrb method. The logp method accepts parameters for class name and method name. The log and logrb methods will generally try to infer this information, but the performance penalty is prohibitive.
2. Avoid using the logrb method. This method leads to inefficient caching of resource bundles and poor performance.
3. Use the isLoggable method to avoid creating data for a logging call that does not get logged. For example:

```
if (logger.isLoggable(Level.FINEST)) {
  String s = dumpComponentState(); // some expensive to compute method
  logger.logp(Level.FINEST, className, methodName, "componentX state
dump:\n{0}", s);
 }
```

The following sample applies to localized messages:

```
// note - generally avoid use of FINE, FINER, FINEST levels for messages to be consistent with
// WebSphere Application Server

String componentName = "com.ibm.websphere.componentX";
String resourceBundleName = "com.ibm.websphere.componentX.Messages";
Logger logger = Logger.getLogger(componentName, resourceBundleName);

// "Convenience" methods - not generally recommended due to lack of class
/ method names
//   - cannot specify message substitution parameters
//   - cannot  specify class and method names
if (logger.isLoggable(Level.SEVERE))
 logger.severe("MSG_KEY_01");

if (logger.isLoggable(Level.WARNING))
 logger.warning("MSG_KEY_01");

if (logger.isLoggable(Level.INFO))
 logger.info("MSG_KEY_01");

if (logger.isLoggable(Level.CONFIG))
 logger.config("MSG_KEY_01");


// log methods are not generally used due to lack of class and method
names
//   - enable use of WebSphere Application Server-specific levels
//   - enable use of message substitution parameters
//   - cannot specify class and method names
if (logger.isLoggable(WsLevel.FATAL))
 logger.log(WsLevel.FATAL, "MSG_KEY_01", "parameter 1");

if (logger.isLoggable(Level.SEVERE))
 logger.log(Level.SEVERE, "MSG_KEY_01", "parameter 1");

if (logger.isLoggable(Level.WARNING))
 logger.log(Level.WARNING, "MSG_KEY_01", "parameter 1");

if (logger.isLoggable(WsLevel.AUDIT))
 logger.log(WsLevel.AUDIT, "MSG_KEY_01", "parameter 1");

if (logger.isLoggable(Level.INFO))
 logger.log(Level.INFO, "MSG_KEY_01", "parameter 1");

if (logger.isLoggable(Level.CONFIG))
 logger.log(Level.CONFIG, "MSG_KEY_01", "parameter 1");

if (logger.isLoggable(WsLevel.DETAIL))
 logger.log(WsLevel.DETAIL, "MSG_KEY_01", "parameter 1");


// logp methods are the way to log
//   - enable use of WebSphere Application Server-specific levels
//   - enable use of message substitution parameters
//   - enable use of class and method names
if (logger.isLoggable(WsLevel.FATAL))
 logger.logp(WsLevel.FATAL, className, methodName, "MSG_KEY_01",
"parameter 1");

if (logger.isLoggable(Level.SEVERE))
 logger.logp(Level.SEVERE, className, methodName, "MSG_KEY_01",
"parameter 1");

if (logger.isLoggable(Level.WARNING))
 logger.logp(Level.WARNING, className, methodName, "MSG_KEY_01",
"parameter 1");
```

```
if (logger.isLoggable(WsLevel.AUDIT))
 logger.logp(WsLevel.AUDIT, className, methodName, "MSG_KEY_01",
"parameter 1");

if (logger.isLoggable(Level.INFO))
 logger.logp(Level.INFO, className, methodName, "MSG_KEY_01",
"parameter 1");

if (logger.isLoggable(Level.CONFIG))
 logger.logp(Level.CONFIG, className, methodName, "MSG_KEY_01",
"parameter 1");

if (logger.isLoggable(WsLevel.DETAIL))
 logger.logp(WsLevel.DETAIL, className, methodName, "MSG_KEY_01",
"parameter 1");


// logrb methods are not generally used due to diminished performance
of switching resource bundles dynamically
//   - enable use of WebSphere Application Server-specific levels
//   - enable use of message substitution parameters
//   - enable use of class and method names
String resourceBundleNameSpecial =
"com.ibm.websphere.componentX.MessagesSpecial";

if (logger.isLoggable(WsLevel.FATAL))
 logger.logrb(WsLevel.FATAL, className, methodName, resourceBundleNameSpecial,
"MSG_KEY_01", "parameter 1");

if (logger.isLoggable(Level.SEVERE))
 logger.logrb(Level.SEVERE, className, methodName, resourceBundleNameSpecial,
"MSG_KEY_01", "parameter 1");

if (logger.isLoggable(Level.WARNING))
 logger.logrb(Level.WARNING, className, methodName, resourceBundleNameSpecial,
"MSG_KEY_01", "parameter 1");

if (logger.isLoggable(WsLevel.AUDIT))
 logger.logrb(WsLevel.AUDIT, className, methodName, resourceBundleNameSpecial,
"MSG_KEY_01", "parameter 1");

if (logger.isLoggable(Level.INFO))
 logger.logrb(Level.INFO, className, methodName, resourceBundleNameSpecial,
"MSG_KEY_01", "parameter 1");

if (logger.isLoggable(Level.CONFIG))
 logger.logrb(Level.CONFIG, className, methodName, resourceBundleNameSpecial,
"MSG_KEY_01", "parameter 1");

if (logger.isLoggable(WsLevel.DETAIL))
 logger.logrb(WsLevel.DETAIL, className, methodName, resourceBundleNameSpecial,
"MSG_KEY_01", "parameter 1");
```

For trace, or content that is not localized, the following sample applies:

```
// note - generally avoid use of FATAL, SEVERE, WARNING, AUDIT,
// INFO, CONFIG, DETAIL levels for trace
// to be consistent with WebSphere Application Server

String componentName = "com.ibm.websphere.componentX";
Logger logger = Logger.getLogger(componentName);

// Entering / Exiting methods are used for non trivial methods
if (logger.isLoggable(Level.FINER))
 logger.entering(className, methodName);
```

```
if (logger.isLoggable(Level.FINER))
 logger.entering(className, methodName, "method param1");

if (logger.isLoggable(Level.FINER))
 logger.exiting(className, methodName);

if (logger.isLoggable(Level.FINER))
 logger.exiting(className, methodName, "method result");


// Throwing method is not generally used due to lack of message - use
logp with a throwable parameter instead
if (logger.isLoggable(Level.FINER))
 logger.throwing(className, methodName, throwable);


// Convenience methods are not generally used due to lack of class
/ method names
//   - cannot specify message substitution parameters
//   - cannot specify class and method names
if (logger.isLoggable(Level.FINE))
 logger.fine("This is my trace");

if (logger.isLoggable(Level.FINER))
 logger.finer("This is my trace");

if (logger.isLoggable(Level.FINEST))
 logger.finest("This is my trace");


// log methods are not generally used due to lack of class and
method names
//   - enable use of WebSphere Application Server-specific levels
//   - enable use of message substitution parameters
//   - cannot specify class and method names
if (logger.isLoggable(Level.FINE))
 logger.log(Level.FINE, "This is my trace", "parameter 1");

if (logger.isLoggable(Level.FINER))
 logger.log(Level.FINER, "This is my trace", "parameter 1");

if (logger.isLoggable(Level.FINEST))
 logger.log(Level.FINEST, "This is my trace", "parameter 1");


// logp methods are the recommended way to log
//   - enable use of WebSphere Application Server-specific levels
//   - enable use of message substitution parameters
//   - enable use of class and method names
if (logger.isLoggable(Level.FINE))
 logger.logp(Level.FINE, className, methodName, "This is my trace",
"parameter 1");

if (logger.isLoggable(Level.FINER))
 logger.logp(Level.FINER, className, methodName, "This is my trace",
"parameter 1");

if (logger.isLoggable(Level.FINEST))
 logger.logp(Level.FINEST, className, methodName, "This is my trace",
"parameter 1");


// logrb methods are not applicable for trace logging because no localization
is involved
```

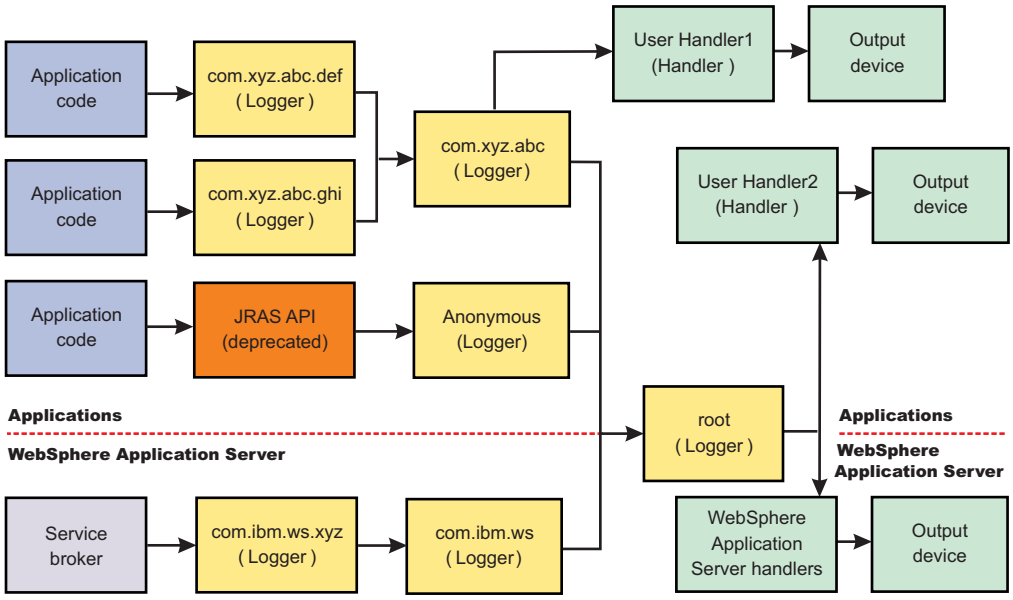
Logger hierarchy

WebSphere Application Server handlers are attached to the Java root logger, which is at the top of the logger hierarchy. As a result, any request from anywhere in the logger tree can be processed by WebSphere Application Server handlers.

With WebSphere Application Server, you can configure the system to do the following:

- Forward all application logging requests to the WebSphere Application Server handlers. This behavior is the default.
- Forward all application logging requests to your own custom handlers. Set the **useParentHandlers** option to `false` on one of your custom loggers, and then attach your handlers to that logger.
- Forward all application logging requests to both WebSphere Application Server handlers, and your custom handlers, but do not forward WebSphere Application Server logging requests to your custom handlers. Set the **useParentHandlers** option to `true` on one of your non-root custom loggers, and then attach your handlers to that logger.`True` is the default setting.
- Forward all WebSphere Application Server logging requests to both WebSphere Application Server handlers, and your custom handlers. WebSphere Application Server logging requests are always forwarded to WebSphere Application Server handlers. To forward WebSphere Application Server requests to your custom handlers, attach your custom handlers to the Java root logger, so that they are at the same level in the hierarchy as the WebSphere Application Server handlers.

The following example shows how these requirements can be met using the Java logging infrastructure.



Creating log resource bundles and message files

Every method that accepts messages localizes those messages. The mechanism for providing localized messages is the resource bundle support provided by the IBM Developer Kit, Java Technology Edition. If you are not familiar with resource bundles as implemented by the Developer Kit, you can get more information from various texts, or by reading the API documentation for the java.util.ResourceBundle, java.util.ListResourceBundle and java.util.PropertyResourceBundle classes, as well as the java.text.MessageFormat class.

The PropertyResourceBundle class is the preferred mechanism to use.

You can forward messages that are written to the internal WebSphere Application Server logs to other processes for display. For example, messages that are displayed on the administrative console, which can

be running in a different location than the server process, can be localized using the *late binding* process. Late binding means that WebSphere Application Server does not localize messages when they are logged, but defers localization to the process that displays the message.

To properly localize the message, the displaying process must have access to the resource bundle where the message text is stored. You must package the resource bundle separately from the application, and install it in a location where the viewing process can access it.

By default, the WebSphere Application Server runtime localizes all the messages when they are logged. This localization eliminates the need to pass a `.jar` file to the application, unless you need to localize in a different location. However, you can use the early binding technique to localize messages as they log. An application that uses early binding must localize the message before logging it. The application looks up the localized text in the resource bundle and formats the message. Use the early binding technique to package the application resource bundles with the application.

To create a resource bundle, perform the following steps.

1. Create a text properties file that lists message keys and the corresponding messages. The properties file must have the following characteristics:
   - Each property in the file is terminated with a line-termination character.
   - If a line contains white space only, or if the first non-white space character of the line is the pound sign symbol (`#`) or exclamation mark (`!`), the line is ignored. The `#` and `!` characters can therefore be used to put comments into the file.
   - Each line in the file, unless it is a comment or consists of white space only, denotes a single property. A backslash (\) is treated as the line-continuation character.
   - The syntax for a property file consists of a key, a separator, and an element. Valid separators include the equal sign (`=`), colon (`:`), and white space ( ).
   - The key consists of all characters on the line from the first non-white space character to the first separator. Separator characters can be included in the key by escaping them with a backslash (\), but doing this process is not recommended, because escaping characters is error prone and confusing. Instead, use a valid separator character that does not display in any keys in the properties file.
   - White space after the key and separator is ignored until the first non-white space character is encountered. All characters remaining before the line-termination character define the element.

   See the Java documentation for the java.util.Properties class for a full description of the syntax and the construction of properties files.
2. Translate the file into localized versions of the file with language-specific file names. For example, a file named `DefaultMessages.properties` can be translated into `DefaultMessages_de.properties` for German and `DefaultMessages_ja.properties` for Japanese.
3. When the translated resource bundles are available, put the bundle in a directory that is part of the application class path.
4. When a message logger is obtained from the log manager, configure it to use a particular resource bundle. Messages logged with the Logger API use this resource bundle when message localization is performed. At run time, the user locale setting determines the properties file from which to extract the message that is specified by a message key, ensuring that the message is delivered in the correct language.
5. If the message loggers msg method is called, a resource bundle name must be explicitly provided.

The application locates the resource bundle based on the file location relative to any directory in the class path. For instance, if the DefaultMessages.properties property resource bundle is located in the *baseDir*`/`*subDir1*`/`*subDir2*`/resources` directory and *baseDir* is in the class path, the name *subdir1*`.`*subdir2*`.resources.DefaultMessage` is passed to the message logger to identify the resource bundle.

Resource bundle logging:

You can create resource bundles in several ways. The best and easiest way is to create a properties file that supports a properties resource bundle. This sample shows how to create such a properties file.

Resource bundle sample

For this sample, four localizable messages are provided. The properties file is created and the key-value pairs are inserted. All the normal properties file conventions and rules apply to this file. In addition, the creator must be aware of other restrictions that are imposed on the values by the Java MessageFormat class. For example, apostrophes must be escaped or they cause a problem. Avoid the use of non-portable characters. WebSphere Application Server does not support the use of extended formatting conventions that the MessageFormat class supports, such as {1, date} or {0,number, integer}.

Assume that the base directory for the application that uses this resource bundle is `baseDir` and that this directory is in the class path. Assume that the properties file is stored in the subdirectory `baseDir` that is not in the class path (for example, `baseDir/subDir1/subDir2/resources`). To allow the messages file to resolve, the `subDir1.subDir2.resources.DefaultMessage` name is used to identify the property resource bundle and is passed to the message logger.

For this sample, the properties file is named `DefaultMessages.properties`.

```
# Contents of the DefaultMessages.properties file
MSG_KEY_00=A message with no substitution parameters.
MSG_KEY_01=A message with one substitution parameter: parm1={0}
MSG_KEY_02=A message with two substitution parameters: parm1={0}, parm2 = {1}
MSG_KEY_03=A message with three parameter: parm1={0}, parm2 = {1}, parm3={2}
```

When the `DefaultMessages.properties` file is created, the file can be sent to a translation center where the localized versions are generated.

Changing the message IDs used in log files

You can change the default format for message IDs in server logs by setting the com.ibm.websphere.logging.messageId.version system property.

In new releases of WebSphere Application Server logging files will be formatted according to a standardized system, but the default runtime behavior is still configured to use the older format, In new releases of WebSphere Application Server the message IDs written to log files will be changed to ensure they do not conflict with other IBM products. The default runtime behavior is still configured to use the older message IDs, deprecated in Version 6.1.

The following is a sample of an entry in a trace.log file using a default message ID. Note that the message ID is PMON0001A

```
[1/26/05 10:17:12:529 EST] 0000000a PMIImpl       A   PMON0001A: PMI is enabled
```

A sample of the same entry using a new message ID follows. Note that the message ID is CWPMI0001A. All new WebSphere Application Server message IDs begin with 'CW'.

```
[1/26/05 10:17:12:529 EST] 0000000a PMIImpl       A   CWPMI0001A: PMI is enabled.
```

If you are using a logging tool that uses the new standardized format, you might want to change the default configuration settings to format the logging output appropriately. You will need to change the configuration for each JVM in the cell if you want the output formatting to be the same across application servers.

To configure logging files to be formatted according to the new system, use the following command in the wsadmin utility:

```
set jvmEntry [$AdminConfig list JavaVirtualMachine]
$AdminConfig create Property $jvmEntry {{name com.ibm.websphere.logging.messageId.version} {value 6} {required false}}
```

Note: You must restart the application server for the changes to take effect. Also, remember to do this for each JVM in the cell for consistent output formatting.

Message IDs written to log files will now be compliant with the new standard.

Converting log files to use IBM unique Message IDs:

The convertlog command creates a new log file with either new or old message IDs substituted in place of the message IDs in the source file.

Prior to Version 6.x, components were assigned message IDs that are not necessarily unique across IBM software products. In Version 6.0, a system property was provided to map the message IDs in output logs to a set of IBM unique message IDs (all WebSphere Application Server message IDs now start with CW) that do not conflict with other IBM software products. The default runtime behavior still uses the old message IDs.

To facilitate the migration of logging tools that are reliant on the old message IDs, the convertlog command is provided to convert the message IDs of log entries from the old standard to the new standard, or the new standard back to the old. By default, the software is configured to use the old message IDs when logging, but you can change the default output with the com.ibm.websphere.logging.messageId.version system property. Read “Changing the message IDs used in log files” on page 31 for more information.

Use the convertlog command to convert the log output:

```
convertlog <source file name> <destination file name> [options]
        options: -newMessageFormat convert message IDs to CCCCCnnnnS format
                  (cannot be used with -m5)
                 -oldMessageFormat convert message IDs to CCCCnnnnS format
                  (cannot be used with -m6)
```

After using the convertlog command you have a new file with message IDs in the chosen format.

convertlog command:

The convertlog command is used to convert the message IDs in log entries from the old standard to the new standard, or the new standard back to the old.

Previous versions of WebSphere Application Server used message IDs that are deprecated in WebSphere Application Server Version 6.1. To facilitate the migration of tools based on the old message IDs, the convertlog command is implemented to translate log files from one message ID standard to the other.

Use the convertlog command as follows:

```
convertlog <source file name> <destination file name> [options]
        options: -newMessageFormat convert message IDs to CCCCCnnnnS format
                  (cannot be used with -m5)
                 -oldMessageFormat convert message IDs to CCCCnnnnS format
                  (cannot be used with -m6)
```

MessageConverter class:

The com.ibm.websphere.logging.MessageConverter class provides a method to convert a message ID at the front of a String into either a new message ID or an old message ID. The direction of the conversion is controlled with the *conversionType* argument.

Use the MessageConverter class with log analysis tools to convert message IDs from earlier versions of WebSphere Application Server into the corresponding message IDs that are used in later releases, or to revert message IDs to an earlier format. See the article “Message reference” on page 107 for list of message ID mappings.

Method

`public static java.lang.String` **`convert`**`(java.lang.String` *`in`*`, short` *`conversionType`*`)`

Parameters

Use the following parameters with the `MessageConverter` class:

Parameter Name	Description
in	The message to convert. The method assumes the message ID is the first part of the supplied message with no leading white space.
conversionType	CONVERSION_TYPE_WASV5_TO_WASV6
	CONVERSION_TYPE_WASV6_TO_WASV5

java.util.logging custom log handlers

There may be occasions when you want to propagate log records to your own log handlers rather than participate in integrated logging.

To use a stand-alone log handler, set the useParentHandlers flag to `false` in your application.

The mechanism for creating a customer handler is the Handler class support that is provided by the IBM Developer Kit, Java Technology Edition. If you are not familiar with handlers, as implemented by the Developer Kit, you can get more information from various texts, or by reading the API documentation for the java.util.logging API.

The following sample shows a custom handler:

```
import java.io.FileOutputStream;
import java.io.PrintWriter;
import java.util.logging.Handler;
import java.util.logging.LogRecord;

/**
 * MyCustomHandler outputs contents to a specified file
 */
public class MyCustomHandler extends Handler {

 FileOutputStream fileOutputStream;
 PrintWriter printWriter;

 public MyCustomHandler(String filename) {
  super();

  // check input parameter
  if (filename == null)
   filename = "mylogfile.txt";

  try {
   // initialize the file
   fileOutputStream = new FileOutputStream(filename);
   printWriter = new PrintWriter(fileOutputStream);
  }
  catch (Exception e) {
   // implement exception handling...
  }
 }

 /* (non-API documentation)
  * @see java.util.logging.Handler#publish(java.util.logging.LogRecord)
  */
```

```
 public void publish(LogRecord record) {
  // ensure that this log record should be logged by this Handler
  if (!isLoggable(record))
   return;

  // Output the formatted data to the file
  printWriter.println(getFormatter().format(record));
 }

 /* (non-API documentation)
  * @see java.util.logging.Handler#flush()
  */
 public void flush() {
  printWriter.flush();
 }

 /* (non-API documentation)
  * @see java.util.logging.Handler#close()
  */
 public void close() throws SecurityException {
  printWriter.close();
 }
}
```

java.util.logging custom filters

A custom filter provides optional, secondary control over what is logged, beyond the control that is provided by the level.

The mechanism for creating a customer filter is the Filter interface support that is provided by the IBM Developer Kit, Java Technology Edition. If you are not familiar with filters, as implemented by the Developer Kit, you can get more information from various texts, or by reading the API documentation the for java.util.logging API.

The following example shows a custom filter:

```
import java.util.Vector;
import java.util.logging.Filter;
import java.util.logging.LogRecord;

/**
 * MyCustomFilter rejects any log records whose Level is not contained in the
 * configured list of Levels.
 */
public class MyCustomFilter implements Filter {

 private Vector acceptableLevels;

 public MyCustomFilter(Vector acceptableLevels) {
  super();
  this.acceptableLevels = acceptableLevels;
 }

 /* (non-API documentation)
  * @see java.util.logging.Filter#isLoggable(java.util.logging.LogRecord)
  */
 public boolean isLoggable(LogRecord record) {
  return (acceptableLevels.contains(record.getLevel()));
 }

}
```

java.util.logging custom formatters

A formatter formats events. Handlers are associated with one or more formatters.

The mechanism for creating a customer formatter is the Formatter class support that is provided by the IBM Developer Kit, Java Technology Edition. If you are not familiar with formatters, as implemented by the Developer Kit, you can get more information from various texts, or by reading the API documentation for the java.util.logging API.

The following example shows a custom formatter:

```
import java.util.Date;
import java.util.logging.Formatter;
import java.util.logging.LogRecord;

/**
 * MyCustomFormatter formats the LogRecord as follows:
 * date   level   localized message with parameters
 */
public class MyCustomFormatter extends Formatter {

 public MyCustomFormatter() {
  super();
 }

 public String format(LogRecord record) {

  // Create a StringBuffer to contain the formatted record
  // start with the date.
  StringBuffer sb = new StringBuffer();

  // Get the date from the LogRecord and add it to the buffer
  Date date = new Date(record.getMillis());
  sb.append(date.toString());
  sb.append(" ");

  // Get the level name and add it to the buffer
  sb.append(record.getLevel().getName());
  sb.append(" ");

  // Get the formatted message (includes localization
  // and substitution of paramters) and add it to the buffer
  sb.append(formatMessage(record));

  return sb.toString();
 }
}
```

Custom handlers, filters, and formatters

In some cases you might want to have your own custom log files. Adding custom handlers, filters, and formatters enables you to customize your logging environment beyond what can be achieved by the configuration of the default WebSphere Application Server logging infrastructure.

The following example demonstrates how to add a new handler to process requests to the com.myCompany subtree of loggers (see “Logger hierarchy” on page 29). The main method in this sample gives an example of how to use the newly configured logger.

```
import java.util.Vector;
import java.util.logging.Filter;
import java.util.logging.Formatter;
import java.util.logging.Handler;
import java.util.logging.Level;
import java.util.logging.Logger;

public class MyCustomLogging {

 public MyCustomLogging() {
  super();
 }
```

```
 public static void initializeLogging() {

  // Get the logger that you want to attach a custom Handler to
  String defaultResourceBundleName = "com.myCompany.Messages";
  Logger logger = Logger.getLogger("com.myCompany", defaultResourceBundleName);

  // Set up a custom Handler (see MyCustomHandler example)
  Handler handler = new MyCustomHandler("MyOutputFile.log");

  // Set up a custom Filter (see MyCustomFilter example)
  Vector acceptableLevels = new Vector();
  acceptableLevels.add(Level.INFO);
  acceptableLevels.add(Level.SEVERE);
  Filter filter = new MyCustomFilter(acceptableLevels);

  // Set up a custom Formatter (see MyCustomFormatter example)
  Formatter formatter = new MyCustomFormatter();

  // Connect the filter and formatter to the handler
  handler.setFilter(filter);
  handler.setFormatter(formatter);

  // Connect the handler to the logger
  logger.addHandler(handler);

  // avoid sending events logged to com.myCompany showing up in WebSphere
  // Application Server logs
  logger.setUseParentHandlers(false);

 }

 public static void main(String[] args) {
  initializeLogging();

  Logger logger = Logger.getLogger("com.myCompany");

  logger.info("This is a test INFO message");
  logger.warning("This is a test WARNING message");
  logger.logp(Level.SEVERE, "MyCustomLogging", "main", "This is a test SEVERE message");
 }
}
```

When the above program is run, the output of the program is written to the MyOutputFile.log file. The content of the log is in the expected log file, as controlled by the custom handler, and is formatted as defined by the custom formatter. The warning message is filtered out, as specified by the configuration of the custom filter. The output is as follows:

```
C:\>type MyOutputFile.log
Sat Sep 04 11:21:19 EDT 2004 INFO This is a test INFO message
Sat Sep 04 11:21:19 EDT 2004 SEVERE This is a test SEVERE message
```

Configuring applications to use Jakarta Commons Logging

Jakarta Commons Logging provides a simple logging interface and thin wrappers for several logging systems. WebSphere Application Server supports Jakarta Commons Logging by providing a logger. The support does not change interfaces defined by Jakarta Commons Logging.

The WebSphere Application Server logger is a thin wrapper for the WebSphere Application Server logging facility. The logger name is *com.ibm.websphere.commons.logging.WsJDK14Logger*. The logger can handle logging objects defined by either of the following:

- Java Logging found in Java Specification Request 47: Logging API Specification
- Common Base Event

A *logging object* is an object that holds logging entry information.

To better understand Jakarta Commons Logging, read Jakarta Commons and the specifications for Java Logging and for Common Base Event. To better understand use of the WebSphere Application Server logger, read “Jakarta Commons Logging.”

WebSphere Application Server provides the Jakarta Commons Logging binary distribution in its `libraries` directory. By default, the product uses the Jakarta Commons Logging LogFactory implementation and JDK14Logger.

For an application to use the WebSphere Application Server logger, the application must provide its own configuration for the logger. To configure an application to use the WebSphere Application Server logger, complete the steps that follow.

1. Examine “Configurations for the WebSphere Application Server logger” on page 40 and determine which configuration best suits your application.
2. Change your application configuration as needed to enable use of the WebSphere Application Server logger.

After the application starts, Jakarta Commons Logging routes the application’s logging output to the WebSphere Application Server logger.

Jakarta Commons Logging

Jakarta Commons Logging provides a simple logging interface and thin wrappers for several logging systems. The logging interface enables application logging to be simple and independent of the logging system that the application uses. You can change the logging implementation for a deployed application without having to change the application logging code. However, the simplicity of the logging interface prevents the application from leveraging all the functionality of the logging systems.

This topic provides the following information about Jakarta Commons Logging in WebSphere Application Server:

Support for Jakarta Commons Logging

WebSphere Application Server supports Jakarta Commons Logging by providing a logger, a thin wrapper for the WebSphere Application Server logging facility. The logger can handle both Java Logging (JSR-47) and Common Base Event logging objects. A *logging object* is an object that holds logging entry information.

The WebSphere Application Server support for Jakarta Commons Logging does not change interfaces defined by Jakarta Commons Logging.

Benefits of support for Jakarta Commons Logging

The WebSphere Application Server support for Jakarta Commons Logging provides the following benefits:

- WebSphere Application Server is pre-configured to use Jakarta Commons Logging.

  All of the functionality of Jakarta Commons Logging is provided for any application or WebSphere Application Server component. Logging calls are routed by default to the underlying WebSphere Application Server logging facility.
- A logger that uses the WebSphere Application Server logging facility.

Applications and components can pass both Java Logging and Common Base Event logging objects to the WebSphere Application Server logger without conversion to strings, providing applications with enhanced logging. Further, Jakarta Commons Logging Logger levels are integrated into WebSphere Application Server administrative facilities.

Overview of the process for using Jakarta Commons Logging

Logging with Jakarta Commons Logging consists of the steps that follow. "Configurations for the WebSphere Application Server logger" on page 40 provides details on configuring your application to use the WebSphere Application Server logger.

1. Obtain an instance of a logger factory.

   To obtain a logger factory, use Jakarta Commons Logging code. You can configure the code to meet your needs. In WebSphere Application Server, Jakarta Commons Logging is configured by default to instantiate the Jakarta Commons Logging default logger factory. Applications or WebSphere Application Server components can provide their own configuration if they use a different logger factory implementation. Applications can use more than one factory.

2. Obtain an instance of a logger.

   To obtain a logger, use code implemented by a logger factory. Configuration of the code is implementation specific.

   The WebSphere Application Server logger implements the methods defined in the logging interface. The logging methods take at least one argument, which can be any Java object. The WebSphere Application Server logger, the WsJDK14Logger logger described in "Classes used to obtain a logger factory and logger," handles the following objects passed into the following logging methods:

   CommonBaseEvent
   Wrapped into CommonBaseEventLogRecord

   CommonBaseEventLogRecord
   Passed without change

   LogRecord
   Passed without change

   Other objects
   Converted to String

   Applications or WebSphere Application Server components can provide their own configuration if they use an implementation of a logger that is not specific to WebSphere Application Server. An application must know what factory is being used in order to configure it.

3. Start your application. Jakarta Commons Logging routes the application's logging output to the designated logger

Classes used to obtain a logger factory and logger

Class name	Description
LogFactory	*LogFactory* is a Jakarta Commons Logging class that implements initialization logic. LogFactory is an abstract class that every logger factory implementation has to extend. It provides static methods for obtaining: • An instance of a factory class • Instances of a logger, using an instance of the factory class LogFactory provides methods for obtaining instances of loggers, although these methods delegate the logger instantiation and configuration to an instance of a logger factory class. Logger factories, once instantiated, are cached on a per context class loader basis. The instances in a cache can be released. This functionality is designed for platform container implementations rather than for applications.

Class name	Description
LogFactoryImpl	*LogFactoryImpl* is a Jakarta Commons Logging concrete class that implements the default logger factory using methods in LogFactory. To use Java Logging, there must always be at least one instance of a logger factory class, even if the application has not explicitly obtained one. If the configuration does not name a logger factory class, LogFactoryImpl is used as the default.
Log	*Log* is a Jakarta Commons Logging interface for loggers. Commons logging loggers have to implement the Log interface. Because the goal of Jakarta Commons Logging is to wrapper any logging system, the Log interface defines a small set of common logging methods. In WebSphere Application Server, WsJDK14Logger implements the Log interface. Logger instantiation and configuration is specific to every logger factory. Logging in WebSphere Application Server uses the default logger factory provided in Jakarta Commons Logging, which keeps instantiated loggers in cache, on a per class loader context basis.
WsJDK14Logger	*WsJDK14Logger* is a WebSphere Application Server class that provides a Jakarta Commons Logging logger by implementing the Log interface. The WsJDK14Logger logger differs from the Java Logging logger in that the WsJDK14Logger logger enables Java Logging or Common Base Event objects to be passed over without converting them into String objects. This prevents any information loss the conversion to String might cause as well as allows the logging output to be more descriptive and precise. In contrast, the Java Logginglogger that is provided in Jakarta Commons Logging converts objects passed into the logging calls to String objects before passing them over to the underlying Java Logging.

Logger level configuration and mapping

Because Jakarta Commons Logging loggers are thin wrappers for specific logging systems, the loggers do not have their own level, but use the level of the logger from the underlying logging system. Although the underlying system can provide methods for changing level, there are no methods for changing level defined on the Log interface, which all Jakarta Commons Logging logger must implement. WsJDK14Logger uses the level of its underlying Java Logging logger.

Following table shows, on the left, the mapping of Jakarta Commons Logging levels within WsJDK14Logger to levels in the WebSphere Application Server implementation of Java Logging. On the right, it shows the levels defined in Java Logging and the level mapping in the Jakarta Commons Logging JDK14Logger to the Java Logging levels.

WsJDK14Logger	Java Logging in WebSphere Application Server	Java Logging	JDK14Logger
Fatal	Fatal		
Error	Severe	Severe	Fatal, Error
Warning	Warning	Warning	Warning
	Audit		
Info	Info	Info	Info
	Config	Config	
	Detail		
Debug	Fine	Fine	Debug
	Finer	Finer	
Trace	Finest	Finest	Trace

The WsJDK14Logger level is synchronized with the underlying Java Logging logger level. WebSphere Application Server administration controls the WsJDK14Logger level.

Configurations for the WebSphere Application Server logger

This topic describes several ways to configure an application to use the WebSphere Application Server logger.

The type of configuration that best suits an application depends upon the following:

- Whether the class loader order setting for the application is `Classes loaded with parent class loader first (Parent First)` or `Classes loaded with application class loader first (Parent Last)`, you can set the class loader delegation mode on a console page. For more details about class load order and delegation, consult the class loading chapter in the *Developing and deploying applications* PDF book
- Whether Jakarta Commons Logging is bundled with the application configuration
- Whether Jakarta Commons Logging is provided within the application

The following tables describe the conditions required to enable an application to use the WebSphere Application Server logger.

Class loader mode is `Parent First` and Jakarta Commons Logging is bundled with the application

Jakarta Commons Logging configuration	LogFactory instance	Log instance	Comments
The application provides the configuration by either of the following: • The properties file `commons-logging.properties` in the application classpath **is not read** by the LogFactory because the parent class loader finds the WebSphere properties file first. • The class name **is read** from the file `META-INF/services/org.apache.commons.logging.LogFactory`	The log factory used is the LogFactory implementation specified in the WebSphere Application Server default configuration, unless the configuration is provided in a META-INF file of the application or module.	The log used is either of the following: • The Log implementation specified in the WebSphere Application Server default configuration • An application-specific Log implementation if an application-specific LogFactory that instantiates a different Log implementation is used.	The **application parent class loader** is the first class loader to load the Jakarta Commons Logging code. The **WebSphere bundle** that supports Jakarta Commons Logging provides the LogFactory static code that looks up the LogFactory configuration attributes. For the static LogFactory code to instantiate the LogFactory instance specified in the application configuration, the LogFactory instance must be on the classpath of the **parent class loader**.
Not provided by the application	The log factory used is the LogFactory implementation specified in the WebSphere default configuration.	The log used is the Log implementation specified in the WebSphere default configuration.	The Jakarta Commons Logging bundled with the application is not used.

Class loader mode is `Parent First` and Jakarta Commons Logging is not bundled with the application

Jakarta Commons Logging configuration	LogFactory instance	Log instance	Comments
The application provides the configuration by either of the following: • The properties file `commons-logging.properties` in the application classpath **is not read** by the LogFactory because the parent class loader finds the WebSphere Application Server properties file first. • The class name **is read** from the file `META-INF/services/org.apache.commons.logging.LogFactory`	The log factory used is the LogFactory implementation specified in the WebSphere Application Server default configuration, unless the configuration is provided in a META-INF file of the application or module.	The log used is either of the following: • The Log implementation specified in the WebSphere Application Server default configuration • An application-specific Log implementation if an application-specific LogFactory that instantiates a different Log implementation is used.	The **application parent class loader** is the first class loader to load the Jakarta Commons Logging code. The **WebSphere bundle** that supports Jakarta Commons Logging provides the LogFactory static code that looks up the LogFactory configuration attributes. For the static LogFactory code to instantiate the LogFactory instance specified in the application configuration, the LogFactory instance must be on the classpath of the **parent class loader**.
Not provided by the application	The log factory used is the LogFactory implementation specified in the WebSphere Application Server default configuration.	The log used is the Log implementation specified in the WebSphere Application Server default configuration.	Same as in the previous row

Class loader mode is `Parent Last` and Jakarta Commons Logging is bundled with the application

Jakarta Commons Logging configuration	LogFactory instance	Log instance	Comments
The application provides the configuration by either of the following: • The properties file `commons-logging.properties` in the application classpath **is read** by the LogFactory because the parent class loader finds the WebSphere Application Server properties file first. • The class name **is read** from the file `META-INF/services/org.apache.commons.logging.LogFactory`	The log factory used is either of the following: • The default Jakarta Commons Logging LogFactory • The LogFactory specified in the application configuration	The log used is the Log implementation specified in the application configuration. If the log factory used is the default Jakarta Commons Logging LogFactory, the Log implementation must be on the classpath of the application class loader.	The **application class loader** is the first class loader to load the Jakarta Commons Logging code. The **application bundle** that supports Jakarta Commons Logging provides the LogFactory static code that looks up the LogFactory configuration attributes. For the static LogFactory code to instantiate the LogFactory instance specified in the application configuration, the LogFactory instance must be on the classpath of the **application class loader**.

Jakarta Commons Logging configuration	LogFactory instance	Log instance	Comments
Not provided by the application	The log factory used is the LogFactory implementation specified in the WebSphere Application Server default configuration.	The log used is the Log implementation specified in the WebSphere Application Server default configuration.	

Class loader mode is `Parent Last` and Jakarta Commons Logging is not bundled with the application

Jakarta Commons Logging configuration	LogFactory instance	Log instance	Comments
The application provides the configuration by either of the following: • The properties file `commons-logging.properties` in the application classpath **is read** by the LogFactory because the parent class loader finds the WebSphere properties file first. • The class name **is read** from the file `META-INF/services/org.apache.commons.logging.LogFactory`	The log factory used is either of the following: • The default Jakarta Commons Logging LogFactory • The LogFactory specified in the application configuration	The log used is the Log implementation specified in the application configuration. If the log factory used is the default Jakarta Commons Logging LogFactory, the Log implementation must be on the classpath of the application class loader.	There is no Jakarta Commons Logging code at the application class loader. Thus, the **WebSphere bundle** that supports Jakarta Commons Logging provides the LogFactory static code that looks up the LogFactory configuration attributes. For the static LogFactory code to instantiate the LogFactory instance specified in the application configuration, the LogFactory instance must be on the classpath of the **parent class loader**.
Not provided by the application	The log factory used is the LogFactory implementation specified in the WebSphere Application Server default configuration.	The log used is the Log implementation specified in the WebSphere Application Server default configuration.	

Programming with the JRas framework

Use the JRas extensions to incorporate message logging and diagnostic trace into WebSphere Application Server applications.

The JRas framework that is described in this task and its sub-tasks is deprecated. However, you can achieve similar results using Java logging.

The JRas extensions allow message logging and diagnostic trace to work with WebSphere Application Server applications. They are based on the stand-alone JRas logging toolkit.

1. Retrieve a reference to the JRas manager.
2. Retrieve message and trace loggers by using methods on the returned manager.

3. Call the appropriate methods on the returned message and trace loggers to create message and trace entries, as appropriate.

JRas logging toolkit

The JRas logging toolkit provides diagnostic information to help the administrator diagnose problems or tune application performance.

Deprecated: The JRas framework that is described in this task and its sub-tasks is deprecated. However, you can achieve similar results using Java logging.

Developing, deploying, and maintaining applications are complex tasks. For example, when a running application encounters an unexpected condition, it might not be able to complete a requested operation. In such a case, you might want the application to inform the administrator that the operation failed and provide information. This action enables the administrator to take the proper corrective action. Those who develop or maintain applications might need to gather detailed information relating to the path of a running application to determine the root cause of a failure that is due to a code bug. The facilities that are used for these purposes are typically referred to as *message logging* and *diagnostic trace*.

Message logging (messages) and diagnostic trace (trace) are conceptually quite similar, but do have important differences. It is important for application developers to understand these differences to use these tools properly. To start with, the following operational definitions of messages and trace are provided.

Message
A message entry is an informational record that is intended for end users, systems administrators and support personnel to view. The text of the message must be clear, concise, and interpretable. Messages are typically localized, meaning that they display in the national language of the end user. Although the destination and lifetime of messages might be configurable, some level of message logging is always enabled in normal system operation. Message logging must be used judiciously due to both performance considerations and the size of the message repository.

Trace A trace entry is an information record that is intended for service engineers or developers to use. This trace record might be considerably more complex, verbose, and detailed than a message entry. Localization support is typically not used for trace entries. Trace entries can be fairly inscrutable, understandable only by the appropriate developer or service personnel. It is assumed that trace entries are not written during normal runtime operation, but might be enabled as needed to gather diagnostic information.

WebSphere Application Server provides a message logging and diagnostic trace API that applications can use. This API is based on the stand-alone JRas logging toolkit, which was developed by IBM. The stand-alone JRas logging toolkit is a collection of interfaces and classes that provide message logging and diagnostic trace primitives. These primitives are not tied to any particular product or platform. The stand-alone JRas logging toolkit provides a limited amount of support, which is typically referred to as *systems management support*, including log file configuration support based on property files.

As designed, the stand-alone JRas logging toolkit does not contain the support that is required for integration into the WebSphere Application Server run time or for use in a Java 2 Platform, Enterprise Edition (J2EE) environment. To overcome these limitations, WebSphere Application Server provides a set of extension classes to address these shortcomings. This collection of extension classes is referred to as the JRas extensions. The JRas extensions do not modify the interfaces that are introduced by the stand-alone JRas logging toolkit, but provide the appropriate implementation classes. The conceptual structure that is introduced by the stand-alone JRas logging toolkit is described in the following section. It is equally applicable to the JRas extensions.

JRas concepts

The section contains a basic overview of important concepts and constructs that are introduced by the stand-alone JRas logging toolkit. This information is not an exhaustive overview of the capabilities of this

logging toolkit, nor is it intended as a detailed discussion of usage or programming paradigms. More detailed information, including code examples, is available in JRas extensions and its subtopics, including in the API documentation for the various interfaces and classes that make up the logging toolkit.

Event types

The stand-alone JRas logging toolkit defines a set of event types for messages and a set of event types for trace. Examples of message types include informational, warning, and error. Examples of trace types include entry, exit, and trace.

Event classes

The stand-alone JRas logging toolkit defines both message and trace event classes.

Loggers

A logger is the primary object with which the user code interacts. Two types of loggers are defined: message loggers and trace loggers. The set of methods on message loggers and trace loggers are different because they provide different functionality. Message loggers create message records only and trace loggers create trace records only. Both types of loggers contain masks that indicate which categories of events the logger processes and which to ignore. Although every JRas logger is defined to contain both a message and trace mask, the message logger uses only the message mask and the trace logger uses the trace mask only. For example, by setting a message logger message mask to the appropriate state, it can be configured to process only error messages and ignore informational and warning messages. Changing the trace mask state of a message logger has no effect.

A logger contains one or more handlers to which it forwards events for further processing. When the user calls a method on the logger, the logger compares the event type that is specified by the caller to its current mask value. If the specified type passes the mask check, the logger creates an event object to capture the information relating to the event that passed to the logger method. This information can include information, such as the names of the class and method which logs the event, a message, and parameters to log, among others. When the logger creates the event object, it forwards the event to all handlers currently registered with the logger.

Methods that are used within the logging infrastructure do not make calls to the logger method. When an application uses an object that extends a thread class, implements the hashCode method, and makes a call to the logging infrastructure from that method, the result is a recursive loop.

Handlers

A handler provides an abstraction over an output device or event consumer. An example is a file handler, which knows how to write an event to a file. The handler also contains a mask that is used to further restrict the categories of events the handler processes. For example, a message logger might be configured to pass both warning and error events, but a handler attached to the message logger might be configured to pass error events only. Handlers also include formatters, which the handler invokes to format the data in the passed event before it is written to the output device.

Formatters

Handlers are configured with formatters, which know how to format events of certain types. A handler can contain multiple formatters, each of which knows how to format a specific class of event. The event object is passed to the appropriate formatter by the handler. The formatter returns formatted output to the handler, which then writes it to the output device.

JRas Extensions

JRas extensions is the collection of implementation classes that support JRas integration into the WebSphere Application Server environment.

JRas extensions

The JRas framework described in this task and its sub-tasks is deprecated. However, you can achieve similar results using Java logging.

The stand-alone JRas logging toolkit defines interfaces and provides a variety of concrete classes that implement these interfaces. Because the stand-alone JRas logging toolkit is developed as a general purpose toolkit, the implementation classes do not contain the configuration interfaces and methods that are necessary for use in the WebSphere Application Server product. In addition, many of the implementation classes are not written appropriately for use in a Java 2 Platform, Enterprise Edition (J2EE) environment. To overcome these shortcomings, WebSphere Application Server provides the appropriate implementation classes that support integration into the WebSphere Application Server environment. The collection of these implementation classes is referred to as the *JRas extensions*.

Usage model

You can use the JRas extensions in three distinct operational modes:

Integrated
: In this mode, message and trace records are written only to logs that are defined and maintained by the WebSphere Application Server run time. This mode is the default mode of operation and is equivalent to the WebSphere Application Server V4.0 mode of operation.

stand-alone
: In this mode, message and trace records are written solely to stand-alone logs that are defined and maintained by the user. You control which categories of events are written to which logs, and the format in which entries are written. You are responsible for configuration and maintenance of the logs. Message and trace entries are not written to WebSphere Application Server runtime logs.

Combined
: In this mode, message and trace records are written to both WebSphere Application Server runtime logs and to stand-alone logs that you must define, control, and maintain. You can use filtering controls to determine which categories of messages and trace are written to which logs.

The JRas extensions are specifically targeted to an integrated mode of operation. The integrated mode of operation can be appropriate for some usage scenarios, but many scenarios are not adequately addressed by these extensions. Many usage scenarios require a stand-alone or combined mode of operation instead. A set of user extension points are defined that support JRas extensions in either a stand-alone or combined mode of operations.

JRas extension classes

WebSphere Application Server provides a base set of implementation classes that are collectively referred to as the *JRas extensions*. Many of these classes provide the appropriate implementations of loggers, handlers, and formatters for use in a WebSphere Application Server environment.

The JRas framework described in this task and its sub-tasks is deprecated. However, you can achieve similar results using Java logging.

The collection of JRas classes is targeted at an integrated mode of operation. If you choose to use the JRas extensions in either stand-alone or combined mode, you can reuse the logger and manager class that are provided by the extensions, but you must provide your own implementations of handlers and formatters.

WebSphere Application Server message and trace loggers

The message and trace loggers that are provided by the stand-alone JRas logging toolkit cannot be directly used in the WebSphere Application Server environment. The JRas extensions provide the appropriate logger implementation classes. Instances of these message and trace logger classes are obtained directly and exclusively from the WebSphere Application Server Manager class. You cannot directly instantiate message and trace loggers. Obtaining loggers in any manner other than directly from the Manager class is not allowed and directly violates the programming model.

The message and trace logger instances that are obtained from the WebSphere Application Server Manager class are subclasses of the RASMessageLogger and RASTraceLogger classes that are provided

by the stand-alone JRas logging toolkit. The RASMessageLogger and RASTraceLogger classes define the set of methods that are directly available. Public methods that are introduced by the JRas extensions logger subclasses cannot be called directly by user code because it is a violation of the programming model.

Loggers are named objects and are identified by name. When the Manager class is called to obtain a logger, the caller is required to specify a name for the logger. The Manager class maintains a name-to-logger instance mapping. Only one instance of a named logger is ever created within the lifetime of a process. The first call to the Manager class with a particular name results in the logger, which is configured by the Manager class. The Manager class caches a reference to the instance, then returns it to the caller. Subsequent calls to the Manager class that specify the same name result in a returned reference to the cached logger. Separate namespaces are maintained for message and trace loggers. You can use a single name obtain both a message logger and a trace logger from the Manager, without ambiguity, and without causing a namespace collision.

In general, loggers have no predefined granularity or scope. A single logger can be used to instrument an entire application. You might determine that having a logger per class is more effective, or the appropriate granularity might be somewhere in between. Partitioning an application into logging domains is determined by the application writer.

The WebSphere Application Server logger classes that are obtained from the Manager class are thread-safe. Although the loggers provided as part of the stand-alone JRas logging toolkit implement the serializable interface, loggers are not serializable. Loggers are stateful objects, tied to a Java virtual machine instance and are not serializable. Attempting to serialize a logger is a violation of the programming model.

Personal or individual logger subclasses are not supported in a WebSphere Application Server environment.

WebSphere Application Server handlers

WebSphere Application Server provides the appropriate handler class that is used to write message and trace events to the WebSphere Application Server run time logs. You cannot configure the WebSphere Application Server handler to write to any other destination. The creation of a WebSphere Application Server handler is a restricted operation and is not available to user code. Every logger that is obtained from the Manager comes preconfigured with an instance of this handler already installed. You can remove the WebSphere Application Server handler from a logger when you want to run in stand-alone mode. When you remove it, you cannot add the WebSphere Application Server handler again to the logger from which it is removed or any other logger. Also, you cannot directly call any method on the WebSphere Application Server handler. Attempting to create an instance of the WebSphere Application Server handler, to call methods on the WebSphere Application Server handler or to add a WebSphere Application Server handler to a logger by user code is a violation of the programming model.

WebSphere Application Server formatters

The WebSphere Application Server handler comes preconfigured with the appropriate formatter for data that is written to WebSphere Application Server logs. The creation of a WebSphere Application Server formatter is a restricted operation and not available to user code. No mechanism exists that allows the user to obtain a reference to a formatter installed in a WebSphere Application Server handler, or to change the formatter a WebSphere Application Server handler is configured to use.

WebSphere Application Server manager

WebSphere Application Server provides a Manager class in the com.ibm.websphere.ras package. All message and trace loggers must be obtained from this Manager class. A reference to the Manager class is obtained by calling the static Manager.getManager method. Message loggers are obtained by calling the

createRASMessageLogger method on the Manager class. Trace loggers are obtained by calling the createRASTraceLogger method on the Manager class.

The manager also supports a *group* abstraction that is useful when dealing with trace loggers. The group abstraction supports multiple, unrelated trace loggers to register as part of a named entity called a *group*. WebSphere Application Server provides the appropriate systems management facilities to manipulate the trace setting of a group, similar to the way the trace settings of an individual trace logger work.

For example, suppose component A consists of 10 classes. Suppose each class is configured to use a separate trace logger. All 10 trace loggers in the component are registered as members of the same group, for example, Component_A_Group. You can turn on trace for a single class, or you can turn on trace for all 10 classes in a single operation using the group name, if you want a component trace. Group names are maintained within the namespace for trace loggers.

JRas framework (deprecated)

Because the JRas extensions classes do not provide the flexibility and behavior that are required for many scenarios, a variety of extension points are defined. You can write your own implementation classes to obtain the required behavior.

Deprecated: The JRas framework described in this topic is deprecated. However, you can achieve similar results using Java logging.

In general, the JRas extensions require you to call the Manager class to obtain a message logger or trace logger. No provision is made for you to provide your own message or trace logger subclasses. In general, user-provided extensions cannot be used to affect the integrated mode of operation. The behavior of the integrated mode of operation is solely determined by the WebSphere Application Server run time and the JRas extensions classes.

Handlers

The stand-alone JRas logging toolkit defines the RASIHandler interface. All handlers must implement this interface. You can write your own handler classes that implement the RASIHandler interface. Directly create instances of user-defined handlers and add them to the loggers that are obtained from the Manager class.

The stand-alone JRas logging toolkit provides several handler implementation classes. These handler classes are inappropriate for use in the Java 2 Platform, Enterprise Edition (J2EE) environment. You cannot directly use or subclass any of the Handler classes that are provided by the stand-alone JRas logging toolkit. Doing so is a violation of the programming model.

Formatters

The stand-alone JRas logging toolkit defines the RASIFormatter interface. All formatters must implement this interface. You can write your own formatter classes that implement the RASIFormatter interface. You can add these classes to a user-defined handler only. WebSphere Application Server handlers cannot be configured to use user-defined formatters. Instead, directly create instances of your formatters and add them to the your handlers appropriately.

As with handlers, the stand-alone JRas logging toolkit provides several formatter implementation classes. Direct use of these formatter classes is not supported.

Message event types

The stand-alone JRas toolkit defines message event types in the RASIMessageEvent interface. In addition, the WebSphere Application Server reserves a range of message event types for future use. The RASIMessageEvent interface defines three types, with values of `0x01`, `0x02`, and `0x04`. The values `0x08`

through 0x8000 are reserved for future use. You can provide your own message event types by extending this interface appropriately. User-defined message types must have a value of 0x1000 or greater.

Message loggers that are retrieved from the Manager class have their message masks set to pass or process all message event types defined in the RASIMessageEvent interface. To process user-defined message types, you must manually set the message logger mask to the appropriate state by user code after the message logger is obtained from the Manager class. WebSphere Application Server does not provide any built-in systems management support for managing message types.

Message event objects

The stand-alone JRas toolkit provides a RASMessageEvent implementation class. When a message logging method is called on the message logger, and the message type is currently enabled, the logger creates and distributes an event of this class to all handlers that are currently registered with that logger.

You can provide your own message event classes, but they must implement the RASIEvent interface. You must directly create instances of such user-defined message event classes. When it is created, pass your message event to the message logger by calling the message logger's fireRASEvent method directly. WebSphere Application Server message loggers cannot directly create instances of user-defined types in response to calling a logging method (msg.message) on the logger. In addition, instances of user-defined message types are never processed by the WebSphere Application Server handler. You cannot create instances of the RASMessageEvent class directly.

Trace event types

The stand-alone JRas toolkit defines trace event types in the RASITraceEvent interface. You can provide your own trace event types by extending this interface appropriately. In such a case, you must ensure that the values for the user-defined trace event types do not collide with the values of the types that are defined in the RASITraceEvent interface.

Trace loggers that are retrieved from the Manager class typically have their trace masks set to reject all types. A different starting state can be specified by using WebSphere Application Server systems management facilities. In addition, you can change the state of the trace mask for a logger at run-time, using WebSphere Application Server systems management facilities.

To process user-defined trace types, the trace logger mask must be manually set to the appropriate state by user code. WebSphere Application Server systems management facilities cannot be used to manage user-defined trace types, either at start time or run time.

Trace event objects

The stand-alone JRas toolkit provides a RASTraceEvent implementation class. When a trace logging method is called on the WebSphere Application Server trace logger and the type is currently enabled, the logger creates and distributes an event of this class to all the handlers that are currently registered with that logger.

You can provide your own trace event classes. Such trace event classes must implement the RASIEvent interface. You must create instances of such user-defined event classes directly. When it is created, pass the trace event to the trace logger by calling the trace logger's fireRASEvent method directly. WebSphere Application Server trace loggers cannot directly create instances of user-defined types in response to calling a trace method (entry, exit, trace) on the trace logger. In addition, instances of user-defined trace types are never processed by the WebSphere Application Server handler. You cannot create instances of the RASTraceEvent class directly.

User defined types, user defined events and WebSphere Application Server

By definition, the WebSphere Application Server handler processed user-defined message or trace types, or user-defined message or trace event classes. Message and trace entries of either a user-defined type or user-defined event class cannot be written to the WebSphere Application Server run-time logs.

JRas programming interfaces for logging (deprecated):

The JRas framework described in this task and its sub-tasks is deprecated. However, you can achieve similar results using Java logging.

General considerations

You can configure the WebSphere Application Server to use Java 2 security to restrict access to protected resources such as the file system and sockets. Because user-written extensions typically access such protected resources, user-written extensions must contain the appropriate security checking calls, using AccessController doPrivileged calls. In addition, the user-written extensions must contain the appropriate policy file. In general, locating user-written extensions in a separate package is a good practice. It is your responsibility to restrict access to the user-written extensions appropriately.

Writing a handler

User-written handlers must implement the RASIHandler interface. The RASIHandler interface extends the RASIMaskChangeGenerator interface, which extends the RASIObject interface. A short discussion of the methods that are introduced by each of these interfaces follows, along with implementation pointers. For more in-depth information on any of the particular interfaces or methods, see the corresponding product API documentation.

RASIObject interface

The RASIObject interface is the base interface for stand-alone JRas logging toolkit classes that are stateful or configurable, such as loggers, handlers, and formatters.

- The stand-alone JRas logging tookit supports rudimentary properties-file based configuration. To implement this configuration support, the configuration state is stored as a set of key-value pairs in a properties file. The public Hashtable getConfig and public void setConfig(Hashtable ht) methods are used to get and set the configuration state. The JRas extensions do not support properties-based configuration. Implement these methods as no-operations. You can implement your own properties-based configuration using these methods.
- Loggers, handlers, and formatters can be named objects. For example, the JRas extensions require the user to provide a name for the loggers that are retrieved from the manager. You can name your handlers. The public String getName and public void setName(String name) methods are provided to get or set the name field. The JRas extensions currently do not call these methods on user handlers. You can implement these methods as you want, including as no operations.
- Loggers, handlers, and formatters can also contain a description field. The public String getDescription and public void setDescription(String desc) methods can be used to get or set the description field. The JRas extensions currently do not use the description field. You can implement these methods as you want, including as no operations.
- The public String getGroup method is provided for use by the RASManager interface. Since the JRas extensions provide their own Manager class, this method is never called. Implement this as a no-operation.

RASIMaskChangeGenerator interface

The RASIMaskChangeGenerator interface is the interface that defines the implementation methods for filtering of events based on a mask state. It is currently implemented by both loggers and handlers. By definition, an object that implements this interface contains both a message mask and a trace mask,

although both need not be used. For example, message loggers contain a trace mask, but the trace mask is never used because the message logger never generates trace events. Handlers, however, can actively use both mask values. For example, a single handler can handle both message and trace events.

- The public long getMessageMask and public void setMessageMask(long mask) methods are used to get or set the value of the message mask. The public long getTraceMask and public void setTraceMask(long mask) methods are used to get or set the value of the trace mask.

In addition, this interface introduces the concept of *calling back* to interested parties when a mask changes state. The callback object must implement the RASIMaskChangeListener interface.

- The public void addMaskChangeListener(RASIMaskChangeListener listener) and public void removeMaskChangeListener(RASIMaskChangeListener listener) methods are used to add or remove listeners to the handler. The public Enumeration getMaskChangeListeners method returns an enumeration over the list of currently registered listeners. The public void fireMaskChangedEvent(RASMaskChangeEvent mc) method is used to call back all the registered listeners to inform them of a mask change event.

For efficiency reasons, the JRas extensions message and trace loggers implement the RASIMaskChangeListener interface. The logger implementations maintain a composite mask in addition to the logger mask. The logger composite mask is formed by logically *or'ing* the appropriate masks of all handlers that are registered to that logger, then *and'ing* the result with the logger mask. For example, the message logger composite mask is formed by or'ing the message masks of all handlers that are registered with that logger, then and'ing the result with the logger message mask.

All handlers are required to properly implement these methods. In addition, when a user handler is instantiated, the logger that is added must be registered with the handler; use the addMaskChangeListener method. When either the message mask or trace mask of the handler is changed, the logger must be called back to inform it of the mask change. With this process, the logger can dynamically maintain the composite mask.

The RASMaskChangedEvent class is defined by the stand-alone JRas logging toolkit. Direct use of that class by user code is supported in this context.

In addition, the RASIMaskChangeGenerator interface introduces the concept of caching the names of all message and trace event classes that the implementing object process. The intent of these methods is to support a management program such as a graphical user interface to retrieve the list of names, introspect the classes to determine the event types that they might possibly process and display the results. The JRas extensions do not ever call these methods, so they can be implemented as no operations.

- The public void addMessageEventClass(String name) and public void removeMessageEventClass(String name) methodscan be called to add or remove a message event class name from the list. The method public Enumeration getMessageEventClasses returns an enumeration over the list of message event class names. Similarly, the public void addTraceEventClass(String name) and public void removeTraceEventClass(String name) methods can be called to add or remove a trace event class name from the list. The public Enumeration getTraceEventClasses method returns an enumeration over the list of trace event class names.

RASIHandler interface

The RASIHandler interface introduces the methods that are specific to the behavior of a handler.

The RASIHandler interface, as provided by the stand-alone JRas logging toolkit, supports handlers that run in either a synchronous or asynchronous mode. In asynchronous mode, events are typically queued by the calling thread and then written by a worker thread. Because spawning of threads is not supported in the WebSphere Application Server environment, it is expected that handlers do not queue or batch events, although this activity is not expressly prohibited.

- The public int getMaximumQueueSize() and public void setMaximumQueueSize(int size) methods create `IllegalStateException` exceptions to manage the maximum queue size. The public int getQueueSize method is provided to query the actual queue size.
- The public int getRetryInterval and public void setRetryInterval(int interval) methods support the notion of error retry, which implies some type of queueing.
- The public void addFormatter(RASIFormatter formatter), public void removeFormatter(RASIFormatter formatter) and public Enumeration getFormatters methods are provided to manage the list of formatters that the handler can be configured with. Different formatters can be provided for different event classes, if appropriate.
- The public void openDevice, public void closeDevice and public void stop methods are provided to manage the underlying device that the handler abstracts.
- The public void logEvent(RASIEvent event) and public void writeEvent(RASIEvent event) methods are provided to pass events to the handler for processing.

Writing a formatter

User-written formatters must implement the RASIFormatter interface. The RASIFormatter interface extends the RASIObject interface. The implementation of the RASIObject interface is the same for both handlers and formatters. A short discussion of the methods that are introduced by the RASIFormatter interface follows. For more in-depth information on the methods introduced by this interface, see the corresponding product API documentation.

RASIFormatter interface
- The public void setDefault(boolean flag) and public boolean isDefault methods are used by the concrete RASHandler classes that are provided by the stand-alone JRas logging toolkit to determine if a particular formatter is the default formatter. Because these RASHandler classes must never be used in a WebSphere Application Server environment, the semantic significance of these methods can be determined by the user.
- The public void addEventClass(String name), public void removeEventClass(String name) and public Enumeration getEventClasses methods are provided to determine which event classes a formatter can use to format. You can provide the appropriate implementations.
- The public String format(RASIEvent event) method is called by handler objects and returns a formatted String representation of the event.

Programming model summary

The programming model that is described in this section builds upon and summarizes some of the concepts already introduced. This section also formalizes usage requirements and restrictions. Use of the WebSphere Application Server JRas extensions in a manner that does not conform to the following programming guidelines is prohibited.

Deprecated: The JRas framework described in this task and its sub-tasks is deprecated. However, you can achieve similar results using Java logging.

You can use the WebSphere Application Server JRas extensions in three distinct operational modes. The programming models concepts and restrictions apply equally across all modes of operation.
- You must not use implementation classes that are provided by the stand-alone JRas logging toolkit directly, unless specifically noted otherwise. Direct usage of those classes is not supported. IBM Support provides no diagnostic aid or bug fixes relating to the direct use of classes that are provided by the stand-alone JRas logging toolkit.
- You must obtain message and trace loggers directly from the Manager class. You cannot directly instantiate loggers.
- You cannot replace the WebSphere Application Server message and trace logger classes.
- You must guarantee that the logger names that are passed to the Manager class are unique, and follow the documented naming constraints. When a logger is obtained from the Manager class, you must not attempt to change the name of the logger by calling the setName method.

- Named loggers can be used more than once. For any given name, the first call to the Manager class results in the Manager class creating a logger that is associated with that name. Subsequent calls to the Manager class that specify the same name result in a returned reference to the existing logger.
- The Manager class maintains a hierarchical namespace for loggers. Use a dot-separated, fully qualified class name to identify any logger. Other than dots or periods, logger names cannot contain any punctuation characters, such as an asterisk (*), a comma (.), an equals sign (=), a colon (:), or quotes.
- Group names must comply with the same naming restrictions as logger names.
- The loggers returned from the Manager class are subclasses of the RASMessageLogger and the RASTraceLogger classes that are provided by the stand-alone JRas logging toolkit. You can call any public method that is defined by the RASMessageLogger and RASTraceLogger classes. You cannot call any public method that is introduced by the provided subclasses.
- If you want to operate in either stand-alone or combined mode, you must provide your own Handler and Formatter subclasses. You cannot use the Handler and Formatter classes that are provided by the stand-alone JRas logging toolkit. User written handlers and formatters must conform to the documented guidelines.
- Loggers that are obtained from the Manager class come with a WebSphere Application Server handler installed. This handler writes message and trace records to logs that are defined by the WebSphere Application Server run time. Manage these logs using the provided systems management interfaces.
- You can programmatically add and remove user-defined handlers from a logger at any time. Multiple additions and removals of user defined handlers are supported. You are responsible for creating an instance of the handler to add, configuring the handler by setting the handler mask value and formatter appropriately, then adding the handler to the logger using the addHandler method. You are responsible for programmatically updating the masks of user-defined handlers, as appropriate.
- You might get a reference to the handler that is installed within a logger by calling the getHandlers method on the logger and processing the results. You must not call any methods on the handler that are obtained in this way. You can remove the WebSphere Application Server handler from the logger by calling the logger removeHandler method, passing in the reference to the WebSphere Application Server handler. When removed, the WebSphere Application Server handler cannot be added again to the logger.
- You can define your own message type. The behavior of user-defined message types and restrictions on their definitions is discussed in Extending the JRas framework.
- You can define your own message event classes. The use of user-defined message event classes is discussed in Extending the JRas framework.
- You can define your own trace types. The behavior of user-defined trace types and restrictions on your definitions is discussed in Extending the JRas framework.
- You can define your own trace event classes. The use of user-defined trace event classes is discussed in Extending the JRas framework.
- You must programmatically maintain the bits in the message and trace logger masks that correspond to any user-defined types. If WebSphere Application Server facilities are used to manage the predefined types, these updates must not modify the state of any of the bits that correspond to those types. If you are assuming ownership responsibility for the predefined types, then you can change all bits of the masks.

JRas messages and trace event types

This topic describes JRas message and trace event types.

Event types

The JRas framework described in this task and its sub-tasks is deprecated. However, you can achieve similar results using Java logging.

The base message and trace event types that are defined by the stand-alone JRas logging toolkit are not the same as the native types that are recognized by the WebSphere Application Server run-time. Instead, the basic JRas types are mapped onto the native types. This mapping can vary by platform or edition. The mapping is discussed in the following section.

Platform message event types

The message event types that are recognized and processed by the WebSphere Application Server runtime are defined in the RASIMessageEvent interface that is provided by the stand-alone JRas logging toolkit. These message types are mapped onto the native message types, as follows.

WebSphere Application Server native type	JRas RASIMessageEvent type
Audit	TYPE_INFO, TYPE_INFORMATION
Warning	TYPE_WARN, TYPE_WARNING
Error	TYPE_ERR, TYPE_ERROR

Platform trace event types

The trace event types that are recognized and processed by the WebSphere Application Server run time are defined in the RASITraceEvent interface that is provided by the stand-alone JRas logging toolkit. The RASITraceEvent interface provides a rich and complex set of types. This interface defines both a simple set of levels, as well as a set of enumerated types.

- For a user who prefers a simple set of levels, the RASITraceEvent interface provides `TYPE_LEVEL1`, `TYPE_LEVEL2`, and `TYPE_LEVEL3`. The implementations provide support for this set of levels. The levels are hierarchical, enabling level 2 also enables level 1, enabling level 3 also enables levels 1 and 2.
- For users who prefer a more complex set of values that can be *OR'd* together, the RASITraceEvent interface provides `TYPE_API`, `TYPE_CALLBACK`, `TYPE_ENTRY_EXIT`, `TYPE_ERROR_EXC`, `TYPE_MISC_DATA`, `TYPE_OBJ_CREATE`, `TYPE_OBJ_DELETE`, `TYPE_PRIVATE`, `TYPE_PUBLIC`, `TYPE_STATIC`, and `TYPE_SVC`.

The trace event types are mapped onto the native trace types as follows:

Mapping WebSphere Application Server trace types to the JRas RASITraceEvent level types.

WebSphere Application Server native type	JRas RASITraceEvent level type
Event	TYPE_LEVEL1
EntryExit	TYPE_LEVEL2
Debug	TYPE_LEVEL3

Mapping WebSphere Application Server trace types to the JRas RASITraceEvent enumerated types.

WebSphere Application Server native type	JRas RASITraceEvent enumerated types
Event	TYPE_ERROR_EXC, TYPE_SVC, TYPE_OBJ_CREATE, TYPE_OBJ_DELETE
EntryExit	TYPE_ENTRY_EXIT, TYPE_API, TYPE_CALLBACK, TYPE_PRIVATE, TYPE_PUBLIC, TYPE_STATIC
Debug	TYPE_MISC_DATA

For simplicity, it is recommended that one or the other of the tracing type methodologies is used consistently throughout the application. If you decide to use the non-level types, choose one type from each category and use those types consistently throughout the application, to avoid confusion.

Message and trace parameters

The various message logging and trace method signatures accept the `Object`, `Object[]` and `Throwable` parameter types. WebSphere Application Server processes and formats the various parameter types as follows:

Primitives

Primitives, such as int and long are not recognized as subclasses of Object type and cannot be directly passed to one of these methods. A primitive value must be transformed to a proper Object type (Integer, Long) before passing as a parameter.

Object

The toString method is called on the object and the resulting String is displayed. Implement the toString method appropriately for any object that is passed to a message logging or trace method. It is the responsibility of the caller to guarantee that the toString method does not display confidential data such as passwords in clear text, and does not cause infinite recursion.

Object[]

The Object[] type is provided for the case when more than one parameter is passed to a message logging or trace method. The toString method is called on each Object in the array. Nested arrays are not handled, that is none of the elements in the Object array belong in an array.

Throwable

The stack trace of the Throwable type is retrieved and displayed.

Array of primitives

An array of primitive, for example, `byte[]`, `int[]`, is recognized as an Object, but is treated somewhat as a second cousin of Object by Java code. In general, avoid arrays of primitives, if possible. If arrays of primitives are passed, the results are indeterminate and can change, depending on the type of array passed, the API used to pass the array, and the release of the product. For consistent results, user code needs to preprocess and format the primitive array into some type of String form before passing it to the method. If such preprocessing is not performed, the following problems can result:

- [B@924586a0b - This message is deciphered as a byte array at location X. This message is typically returned when an array is passed as a member of an Object[] type and results from calling the toString method on the byte[] type.
- Illegal trace argument : array of long. This response is typically returned when an array of primitives is passed to a method taking an Object.
- 01040703: The hex representation of an array of bytes. Typically this problem can occur when a byte array is passed to a method taking a single Object. This behavior is subject to change and cannot be relied on.
- ″1″ ″2″: The String representation of the members of an int[] type formed by converting each element to an integer and calling the toString method on the integers. This behavior is subject to change and cannot be relied on.
- [Ljava.lang.Object;@9136fa0b : An array of objects. Typically this response is seen when an array containing nested arrays is passed.

Controlling message logging

Writing a message to a WebSphere Application Server log requires that the message type passes three levels of filtering or screening:

1. The message event type must be one of the message event types that is defined in the RASIMessageEvent interface.
2. Logging of that message event type must be enabled by the state of the message logger mask.
3. The message event type must pass any filtering criteria that is established by the WebSphere Application Server run-time.

When a WebSphere Application Server logger is obtained from the Manager class, the initial setting of the mask forwards all native message event types to the WebSphere Application Server handler. It is possible to control what messages get logged by programmatically setting the state of the message logger mask.

Some editions of the product support user specified message filter levels for a server process. When such a filter level is set, only messages at the specified severity levels are written to WebSphere Application Server. Message types that pass the mask check of the message logger can be filtered out by WebSphere Application Server.

Control tracing

Each edition of the product provides a mechanism for enabling or disabling trace. The various editions can support static trace enablement (trace settings are specified before the server is started), dynamic trace enablement (trace settings for a running server process can be dynamically modified), or both.

Writing a trace record to a WebSphere Application Server requires that the trace type passes three levels of filtering or screening:
1. The trace event type must be one of the trace event types that is defined in the RASITraceEvent interface.
2. Logging of that trace event type must be enabled by the state of the trace logger mask.
3. The trace event type must pass any filtering criteria that is established by the WebSphere Application Server run-time.

When a logger is obtained from the Manager class, the initial setting of the mask is to suppress all trace types. The exception to this rule is the case where the WebSphere Application Server run time supports static trace enablement and a non-default startup trace state for that trace logger is specified. Unlike message loggers, the WebSphere Application Server can dynamically modify the trace mask state of a trace logger. WebSphere Application Server only modifies the portion of the trace logger mask that corresponds to the values that are defined in the RASITraceEvent interface. WebSphere Application Server does not modify undefined bits of the mask that might be in use for user-defined types.

When the dynamic trace enablement feature that is available on some platforms is used, the trace state change is reflected both in the application server run time and the trace mask of the trace logger. If user code programmatically changes the bits in the trace mask corresponding to the values that are defined by in the RASITraceEvent interface, the mask state of the trace logger and the run time state become unsynchronized and unexpected results occur. Therefore, programmatically changing the bits of the mask corresponding to the values that are defined in the RASITraceEvent interface is not supported.

Instrumenting an application with JRas extensions

You can create an application using JRas extensions.

The JRas framework that is described in this task and its sub-tasks is deprecated. However, you can achieve similar results using Java logging.

To create an application using the WebSphere Application Server JRas extensions, perform the following steps:
1. Determine the mode for the extensions: integrated, stand-alone, or combined.
2. If the extensions are used in either stand-alone or combined mode, create the necessary handler and formatter classes.
3. If localized messages are used by the application, create a resource bundle.
4. In the application code, get a reference to the Manager class and create the manager and logger instances.
5. Insert the appropriate message and trace logging statements in the application.

Creating JRas resource bundles and message files

The WebSphere Application Server message logger provides the message and msg methods so the user can log localized messages. In addition, the message logger provides the textMessage method to log messages that are not localized. Applications can use either or both, as appropriate.

The JRas framework that is described in this task and its sub-tasks is deprecated. However, you can achieve similar results using Java logging.

The mechanism for providing localized messages is the resource bundle support that is provided by the IBM Developer Kit, Java Technology Edition. If you are not familiar with resource bundles as implemented by the Developer Kit, you can get more information from various texts, or by reading the API documentation for the java.util.ResourceBundle, java.util.ListResourceBundle and java.util.PropertyResourceBundle classes, as well as the java.text.MessageFormat class.

The PropertyResourceBundle class is the preferred mechanism to use. In addition, note that the JRas extensions do not support the extended formatting options such as {1, date} or {0, number, integer} that are provided by the MessageFormat class.

You can forward messages that are written to the internal WebSphere Application Server logs to other processes for display. For example, messages that are displayed on the administrative console, which can be running in a different location than the server process, can be localized using the *late binding* process. Late binding means that WebSphere Application Server does not localize messages when they are logged, but defers localization to the process that displays the message.

To properly localize the message, the displaying process must have access to the resource bundle where the message text is stored. You must package the resource bundle separately from the application, and install it in a location where the viewing process can access it. If you do not want to take these steps, you can use the early binding technique to localize messages as they are logged.

The two techniques are described as follows:

Early binding

The application must localize the message before logging it. The application looks up the localized text in the resource bundle and formats the message. When formatting is complete, the application logs the message using the textMessage method. Use this technique to package the application resource bundles with the application.

Late binding

The application can choose to have the WebSphere Application Server run time localize the message in the process where it displays. Using this technique, the resource bundles are packaged in a stand-alone `.jar` file, separately from the application. You must then install the resource bundle `.jar` file on every machine in the installation from which an administrative console or log viewing program might be run. You must install the `.jar` file in a directory that is part of the extensions class path. In addition, if you forward logs to IBM service, you must also forward the `.jar` file that contains the resource bundles.

To create a resource bundle, perform the following steps.

1. Create a text properties file that lists message keys and the corresponding messages. The properties file must have the following characteristics:
   - Each property in the file is terminated with a line-termination character.
   - If a line contains only white space, or if the first non-white space character of the line is the number sign symbol (#) or exclamation mark (!), the line is ignored. The # and ! characters can therefore be used to put comments into the file.
   - Each line in the file, unless it is a comment or consists only of white space, denotes a single property. A backslash (\) is treated as the line-continuation character.
   - The syntax for a property file consists of a key, a separator, and an element. Valid separators include the equal sign (=), colon (:), and white space ( ).
   - The key consists of all characters on the line from the first non-white space character to the first separator. Separator characters can be included in the key by escaping them with a backslash (\), but using this approach is not recommended because escaping characters is error prone and confusing. Instead, use a valid separator character that does not display in any keys in the properties file.
   - White space after the key and separator is ignored until the first non-white space character is encountered. All characters that remain before the line-termination character define the element.

See the Java documentation for the java.util.Properties class for a full description of the syntax and construction of properties files.

2. Translate the file into localized versions of the file with language-specific file names for example, the DefaultMessages.properties file can be translated into `DefaultMessages_de.properties` for German and `DefaultMessages_ja.properties` for Japanese.
3. When the translated resource bundles are available, write them to a system-managed persistent storage medium. Resource bundles are used to convert the messages into the requested national language and locale.
4. When a message logger is obtained from the JRas manager, configure the logger to use a particular resource bundle. Messages logged through the `message` API use this resource bundle when message localization is performed. At run time, the user's locale setting is used to determine the properties file from which to extract the message that is specified by a message key, ensuring that the message is delivered in the correct language.
5. If the message loggers msg method is called, explicitly identify a resource bundle name.

The application locates the resource bundle based on the file location relative to any directory in the class path. For instance, if the DefaultMessages.properties property resource bundle is in the `baseDir/subDir1/subDir2/resources` directory and `baseDir` is in the class path, the name *subdir1.subdir2*.resources.DefaultMessage is passed to the message logger to identify the resource bundle.

JRas resource bundles:

You can create resource bundles in several ways. The best and easiest way is to create a properties file that supports a PropertiesResourceBundle resource bundle. This sample shows how to create such a properties file.

Resource bundle sample

The JRas framework described in this task and its sub-tasks is deprecated. However, you can achieve similar results using Java logging.

For this sample, four localizable messages are provided. The properties file is created and the key-value pairs are inserted into it. All the normal properties files conventions and rules apply to this file. In addition, the creator must be aware of other restrictions that are imposed on the values by the Java MessageFormat class. For example, apostrophes must be escaped or they cause a problem. Avoid the use of non-portable characters. WebSphere Application Server does not support the use of extended formatting conventions that the MessageFormat class supports, such as {1, date} or {0, number, integer}.

Assume that the base directory for the application that uses this resource bundle is `baseDir` and that this directory is in the class path. Assume that the properties file is stored in the subdirectory `baseDir` that is not in the class path (*baseDir/subDir1/subDir2/resources*). To allow the messages file to resolve, the subDir1.subDir2.resources.DefaultMessage name is used to identify the PropertyResourceBundle resource bundle and is passed to the message logger.

For this sample, the properties file is named `DefaultMessages.properties`:

```
# Contents of the DefaultMessages.properties file
MSG_KEY_00=A message with no substitution parameters.
MSG_KEY_01=A message with one substitution parameter: parm1={0}
MSG_KEY_02=A message with two substitution parameters: parm1={0}, parm2 = {1}
MSG_KEY_03=A message with three substitution parameters: parm1={0}, parm2 = {1}, parm3={2}
```

When the `DefaultMessages.properties` file is created, the file can be sent to a translation center where the localized versions are generated.

JRas manager and logger instances

You can use the JRas extensions in integrated, stand-alone, or combined mode. Configuration of the application varies depending on the mode of operation, but use of the loggers to log message or trace entries is identical in all modes of operation.

Deprecated: The JRas framework described in this task and its sub-tasks is deprecated. However, you can achieve similar results using Java logging.

Integrated mode is the default mode of operation. In this mode, message and trace events are sent to the WebSphere Application Server logs.

In the combined mode, message and trace events are logged to both WebSphere Application Server and user-defined logs.

In the stand-alone mode, message and trace events are logged only to user-defined logs.

Using the message and trace loggers

Regardless of the mode of operation, the use of message and trace loggers is the same.

Using a message logger

The message logger is configured to use the DefaultMessages resource bundle. Message keys must be passed to the message loggers if the loggers are using the message API.

```
msgLogger.message(RASIMessageEvent.TYPE_WARNING, this,
     methodName, "MSG_KEY_00");
... msgLogger.message(RASIMessageEvent.TYPE_WARN, this,
     methodName, "MSG_KEY_01", "some string");
```

If message loggers use the msg API, you can specify a new resource bundle name.

```
msgLogger.msg(RASIMessageEvent.TYPE_ERR, this, methodName,
                  "ALT_MSG_KEY_00", "alternateMessageFile");
```

You can also log a text message. If you are using the textMessage API, no message formatting is done.

```
msgLogger.textMessage(RASIMessageEvent.TYPE_INFO, this, methodName,"String and Integer",
"A String", new Integer(5));
```

Using a trace logger

Because trace is normally disabled, guard trace methods for performance reasons.

```
private void methodX(int x, String y, Foo z)
{
   // trace an entry point. Use the guard to make sure tracing is enabled.
Do this checking before you gather parameters to  trace.
   if (trcLogger.isLoggable(RASITraceEvent.TYPE_ENTRY_EXIT) {
        // I want to trace three parameters, package them up in an Object[]
        Object[] parms = {new Integer(x), y, z};
        trcLogger.entry(RASITraceEvent.TYPE_ENTRY_EXIT, this, "methodX", parms);
  }
... logic
  // a debug or verbose trace point
  if (trcLogger.isLoggable(RASITraceEvent.TYPE_MISC_DATA) {
        trcLogger.trace(RASITraceEvent.TYPE_MISC_DATA, this, "methodX" "reached here");
  }
  ...
  // Another classification of trace event. An important state change is
 detected, so a different trace type is used.
  if (trcLogger.isLoggable(RASITraceEvent.TYPE_SVC) {
     trcLogger.trace(RASITraceEvent.TYPE_SVC, this, "methodX", "an important event");
```

```
  }
 ...
  // ready to exit method, trace. No return value to trace
    if (trcLogger.isLoggable(RASITraceEvent.TYPE_ENTRY_EXIT)) {
        trcLogger.exit(RASITraceEvent.TYPE_ENTRY_EXIT, this, "methodX");
   }
}
```

Setting up for integrated JRas operation

Use JRas operations in integrated mode to send trace events and logging messages to only WebSphere Application Server logs.

The JRas framework described in this task and its sub-tasks is deprecated. However, you can achieve similar results using Java logging.

In the integrated mode of operation, message and trace events are sent to WebSphere Application Server logs. This approach is the default mode of operation.

1. Import the requisite JRas extensions classes:

   ```
   import com.ibm.ras.*;
   import com.ibm.websphere.ras.*;
   ```

2. Declare logger references:

   ```
   private RASMessageLogger msgLogger = null;
   private RASTraceLogger trcLogger = null;
   ```

3. Obtain a reference to the Manager class and create the loggers. Because loggers are named singletons, you can do this activity in a variety of places. One logical candidate for enterprise beans is the ejbCreate method. For example, for the myTestBean enterprise bean, place the following code in the ejbCreate method:

   ```
   com.ibm.websphere.ras.Manager mgr = com.ibm.websphere.ras.Manager.getManager();
   msgLogger = mgr.createRASMessageLogger("Acme", "WidgetCounter", "RasTest",
            myTestBean.class.getName());



   // Configure the message logger to use the message file that is created
   // for this application.
   msgLogger.setMessageFile("acme.widgets.DefaultMessages");
   trcLogger = mgr.createRASTraceLogger("Acme", "Widgets", "RasTest",
            myTestBean.class.getName());
   mgr.addLoggerToGroup(trcLogger, groupName);
   ```

Setting up for combined JRas operation

Use JRas operation in combined mode to output trace data and logging messages to both WebSphere Application Server and user-defined logs.

The JRas framework described in this task and its sub-tasks is deprecated. However, you can achieve similar results using Java logging.

In combined mode, messages and trace are logged to both WebSphere Application Server logs and user-defined logs. The following sample assumes that:

- You wrote a user-defined handler named SimpleFileHandler and a user-defined formatter named SimpleFormatter.
- You are not using user-defined types or events.

1. Import the requisite JRas extensions classes:

   ```
   import com.ibm.ras.*;
   import com.ibm.websphere.ras.*;
   ```

2. Import the user handler and formatter:

   ```
   import com.ibm.ws.ras.test.user.*;
   ```

3. Declare the logger references:

```
private RASMessageLogger msgLogger = null;
  private RASTraceLogger trcLogger = null;
```

4. Obtain a reference to the Manager class, create the loggers, and add the user handlers. Because loggers are named singletons, you can obtain a reference to the loggers in a number of places. One logical candidate for enterprise beans is the ejbCreate method. Make sure that multiple instances of the same user handler are not accidentally inserted into the same logger. Your initialization code must support this approach. The following sample is a message logger sample. The procedure for a trace logger is similar.

```
com.ibm.websphere.ras.Manager mgr = com.ibm.websphere.ras.Manager.getManager();
    msgLogger = mgr.createRASMessageLogger("Acme", "WidgetCounter", "RasTest",
           myTestBean.class.getName());
    // Configure the message logger to use the message file defined
    // in the ResourceBundle sample.
    msgLogger.setMessageFile("acme.widgets.DefaultMessages");

   // Create the user handler and formatter. Configure the formatter,
   // then add it to the handler.
   RASIHandler handler = new SimpleFileHandler("myHandler", "FileName");
   RASIFormatter formatter = new SimpleFormatter("simple formatter");
    formatter.addEventClass("com.ibm.ras.RASMessageEvent");
    handler.addFormatter(formatter);

   // Add the Handler to the logger. Add the logger to the list of the
   //handlers listeners, then set the handlers
   // mask, which updates the loggers composite mask appropriately.
   // WARNING - there is an order dependency here that must be followed.
   msgLogger.addHandler(handler);
   handler.addMaskChangeListener(msgLogger);
   handler.setMessageMask(RASIMessageEvent.DEFAULT_MESSAGE_MASK);
```

Setting up for stand-alone JRas operation

You can configure JRas operations to output trace data and logging messages to only user-defined locations.

The JRas framework described in this task and its sub-tasks is deprecated. However, you can achieve similar results using Java logging.

In stand-alone mode, messages and traces are logged only to user-defined logs. The following sample assumes that:

- You have a user-defined handler named SimpleFileHandler and a user-defined formatter named SimpleFormatter.
- You are not using user-defined types of events.

1. Import the requisite JRas extensions classes:

```
import com.ibm.ras.*;
import com.ibm.websphere.ras.*;
```

2. Import the user handler and formatter:

```
import com.ibm.ws.ras.test.user.*;
```

3. Declare the logger references:

```
private RASMessageLogger msgLogger = null;
  private RASTraceLogger trcLogger = null;
```

4. Obtain a reference to the Manager class, create the loggers, and add the user handlers. Because loggers are named singletons, you can obtain a reference to the loggers in a number of places. One logical candidate for enterprise beans is the ejbCreate method. Make sure that multiple instances of the same user handler are not accidentally inserted into the same logger. Your initialization code must support this approach. The following sample is a message logger sample. The procedure for a trace logger is similar.

```
com.ibm.websphere.ras.Manager mgr = com.ibm.websphere.ras.Manager.getManager();
    msgLogger = mgr.createRASMessageLogger("Acme", "WidgetCounter", "RasTest",
             myTestBean.class.getName());
    // Configure the message logger to use the message file that is defined in
    //the ResourceBundle sample.
    msgLogger.setMessageFile("acme.widgets.DefaultMessages");

    // Get a reference to the Handler and remove it from the logger.
    RASIHandler aHandler = null;
    Enumeration enum = msgLogger.getHandlers();
    while (enum.hasMoreElements()) {
         aHandler = (RASIHandler)enum.nextElement();
         if (aHandler instanceof WsHandler)
              msgLogger.removeHandler(wsHandler);
    }

    // Create the user handler and formatter. Configure the formatter,
    // then add it to the handler.
    RASIHandler handler = new SimpleFileHandler("myHandler", "FileName");
    RASIFormatter formatter = new SimpleFormatter("simple formatter");
    formatter.addEventClass("com.ibm.ras.RASMessageEvent");
    handler.addFormatter(formatter);

   // Add the Handler to the logger. Add the logger to the list of the
   // handlers listeners, then set the handlers
   // mask, which will update the loggers composite mask appropriately.
   // WARNING - there is an order dependency here that must be followed.
   msgLogger.addHandler(handler);
   handler.addMaskChangeListener(msgLogger);
   handler.setMessageMask(RASIMessageEvent.DEFAULT_MESSAGE_MASK);
```

Configuring logging properties using the administrative console

Use this task to browse or change the properties of Java logging.

Before applications can log diagnostic information, you need to specify how you want the server to handle log output, and what level of logging you require. Using the administrative console, you can:

- Enable or disable a particular log, specify where log files are stored and how many log files are kept.
- Specify the level of detail in a log, and specify a format for log output.
- Set a log level for each logger.

You can change the log configuration statically or dynamically. Static configuration changes affect applications when you start or restart the application server. Dynamic or run time configuration changes apply immediately.

When a log is created, the level value for that log is set from the configuration data. If no configuration data is available for a particular log name, the level for that log is obtained from the parent of the log. If no configuration data exists for the parent log, the parent of that log is checked, and so on up the tree, until a log with a non-null level value is found. When you change the level of a log, the change is propagated to the children of the log, which recursively propagates the change to their children, as necessary.

To configure loggers and log handlers for Java logging, use the administrative console to complete the following steps:

1. Set the logging levels for your logs:
   a. In the navigation pane, click **Servers** > **Application Servers**.
   b. Click the name of the server that you want to work with.
   c. Under Troubleshooting, click **Logging and tracing**.
   d. Click **Change Log Detail levels**.

e. To make a static change to the configuration, click the **Configuration** tab. A list of well-known components, packages, and groups is displayed. To change the configuration dynamically, click the **Runtime** tab. The list of components, packages, and groups displays all the components that are currently registered on the running server.

f. Select a component, package, or group to set a logging level.

g. Click **Apply**.

h. Click **OK**.

2. To have static configuration changes take effect, stop then restart the application server.

Log level settings

Use this topic to configure and manage log level settings.

Using log levels you can control which events are processed by Java logging. When you change the level for a logger, the change is propagated to the children of the logger.

Change Log Detail Levels

Enter a log detail level that specifies the components, packages, or groups to trace. The log detail level string must conform to the specific grammar described in this topic. You can enter the log detail level string directly, or generate it using the graphical trace interface.

If you select the Configuration tab, a static list of well-known components, packages, and groups is displayed. This list might not be exhaustive.

If you select the Runtime tab, the list of components, packages, and group are displayed with all the components that are registered on the running application server and in the static list.

The format of the log detail level specification is:

```
<component> = <level>
```

where <component> is the component for which to set a log detail level, and <level> is one of the valid logger levels (off, fatal, severe, warning, audit, info, config, detail, fine, finer, finest, all). Separate multiple log detail level specifications with colons (:).

Components correspond to Java packages and classes, or to collections of Java packages. Use an asterisk (*) as a wildcard to indicate components that include all the classes in all the packages that are contained by the specified component. For example:

***** Specifies all traceable code running in the application server, including the product system code and customer code.

com.ibm.ws.*
Specifies all classes with the package name beginning with com.ibm.ws.

com.ibm.ws.classloader.JarClassLoader
Specifies the JarClassLoader class only.

An error can occur when setting a log detail level specification from the administrative console if selections are made from both the Groups and Components lists. In some cases, the selection made from one list is lost when adding a selection from the other list. To work around this problem, enter the log detail level specification directly into the log detail level entry field.

Select a component or group to set a log detail level. The table following lists the valid levels for application servers at WebSphere Application Server Version 6 and later, and the valid logging and trace levels for earlier versions:

Version 6 logging level	Logging level before Version 6	Trace level before Version 6	Content / Significance

Off	Off	All disabled*	Logging is turned off. * In Version 6, a trace level of All disabled turns off trace, but does not turn off logging. Logging is enabled from the Info level.
Fatal	Fatal	-	Task cannot continue and component, application, and server cannot function.
Severe	Error	-	Task cannot continue but component, application, and server can still function. This level can also indicate an impending fatal error.
Warning	Warning	-	Potential error or impending error. This level can also indicate a progressive failure (for example, the potential leaking of resources).
Audit	Audit	-	Significant event affecting server state or resources
Info	Info	-	General information outlining overall task progress
Config	-	-	Configuration change or status
Detail	-	-	General information detailing subtask progress
Fine	-	Event	Trace information - General trace + method entry, exit, and return values
Finer	-	Entry/Exit	Trace information - Detailed trace
Finest	-	Debug	Trace information - A more detailed trace that includes all the detail that is needed to debug problems
All		All enabled	All events are logged. If you create custom levels, All includes those levels, and can provide a more detailed trace than finest.

When you enable a logging level in Version 6.0 or above, you are also enabling all of the levels with higher severity. For example, if you set the logging level to warning on your Version 6.x application server, then warning, severe and fatal events are processed.

Trace information, which are events at the Fine, Finer and Finest levels, can be written only to the trace log. Therefore, if you do not enable diagnostic trace, setting the log detail level to Fine, Finer, or Finest will not have an effect on the data that is logged.

HTTP error and NCSA access log settings

Use this page to configure an HTTP error log and National Center for Supercomputing Applications (NCSA) access logs for an HTTP transport channel. The HTTP error log contains HTTP errors. The level of error logging that occurs is dependent on the value that is selected for the Error log level field.

To view this administrative console page, click **Application servers >** *server name* **> HTTP error and NCSA access logging**.

The NCSA access log contains a record of all inbound client requests that the HTTP transport channel handles. All of the messages that are contained in these logs are in NCSA format.

After you configure the HTTP error log and the NCSA access logs, make sure that the Enable NCSA access logging field is selected for the HTTP channels for which you want logging to occur. To view the settings for an HTTP channel, click **Servers > Application Servers >** *server* **> Web Container Transport Chains > HTTP Inbound Channel**.

Enable service at server startup

When selected, either an NCSA access log or an HTTP error log, or both are initialized when the server starts.

Enable access logging

When selected, a record of inbound client requests that the HTTP transport channel handles is kept in the NCSA access log.

Access log file path

Indicates the directory path and name of the NCSA access log. Standard variable substitutions, such as $(SERVER_LOG_ROOT), can be used when specifying the directory path.

Access log maximum size

Indicates the maximum size, in megabytes, of the NCSA access log file. When this size is reached, the *logfile_name*.1 archive log is created. However, every time that the original log file overflows this archive file, the file is overwritten with the most current version of the original log file.

NCSA access log format

Indicates that the NCSA format is used when logging client access information. If Common is selected, the log entries contain the requested resource and a few other pieces of information, but does not contain referral, user agent, or cookie information. If Combined is selected, referral, user agent, and or cookie information is included.

Enable error logging

When selected, HTTP errors that occur while the HTTP channel processes client requests are recorded in the HTTP error log.

Error log file path

Indicates the directory path and the name of the HTTP error log. Standard variable substitutions, such as $(SERVER_LOG_ROOT), can be used when specifying the directory path.

Error log maximum size

Indicates the maximum size, in megabytes, of the HTTP error log file. When this size is reached, the *logfile_name*.1 archive log is created. However, every time that the original log file overflows this archive file, this file is overwritten with the most current version of the original log file.

Error log level

Indicates the type of error messages that are included in the HTTP error log.

You can select:

Critical
Only critical failures that stop the Application Server from functioning properly are logged.

Error The errors that occur in response to clients are logged. These errors require Application Server administrator intervention if they result from server configuration settings.

Warning
Information on general errors, such as socket exceptions that occur while handling client requests, are logged. These errors do not typically require Application Server administrator intervention.

Information
The status of the various tasks that are performed while handling client requests is logged.

Debug
More verbose task status information is logged. This level of logging is not intended to replace RAS logging for debugging problems, but does provide a steady status report on the progress of individual client requests. If this level of logging is selected, you must specify a large enough log file size in the Error log maximum size field to contain all of the information that is logged.

The Common Base Event in WebSphere Application Server

This topic describes how WebSphere Application Server takes advantage of the Common Base Events.

An application creates an event object whenever something happens that either needs to be recorded for later analysis or which might require the trigger of additional work. An *event* is a structured notification that reports information that is related to a situation. An event reports three kinds of information:
- The situation: What happened
- The identity of the affected component: For example, the server that shut down
- The identity of the component that is reporting the situation, which might be the same as the affected component

The application that creates the event object is called the *event source*. Event sources can use a common structure for the event. The accepted standard for such a structure is called the *Common Base Event*. The Common Base Event is an XML document that is defined as part of the autonomic computing initiative. The Common Base Event defines common fields, the values they can take, and the exact meanings of these values.

The Common Base Event model is a standard that defines a common representation of events that is intended for use by enterprise management and business applications. This standard, which is developed by the IBM Autonomic Computing Architecture Board, supports encoding of logging, tracing, management, and business events using a common XML-based format. This format makes it possible to correlate different types of events that originate from different applications. For more information about the Common Base Event model, see the Common Base Event specification (*Canonical Situation Data Format: The Common Base Event V1.0.1*). The common event infrastructure currently supports Version 1.0.1 of the specification.

The basic concept behind the Common Base Event model is the *situation*. A situation can be anything that happens anywhere in the computing infrastructure, such as a server shutdown, a disk-drive failure, or a failed user login. The Common Base Event model defines a set of standard situation types that accommodate most of the situations that might arise (for example, StartSituation and CreateSituation).

The Common Base Event contains all of the information that is needed by the consumers to understand the event. This information includes data about the runtime environment, the business environment, and the instance of the application object that created the event.

For complete details on the Common Base Event format, see the XML schema that is included in the Common Base Event specification document, at ftp://www6.software.ibm.com/software/developer/library/ac-toolkitdg.pdf .

Types of problem determination events

Problem determination involves multiple types of data, including at least two different classes of event data, log events, and diagnostic events.

Log events, which are also referred to as *message events*, are typically emitted by components of a business application during normal deployment and operations. Log events might identify problems, but these events are also normally available and emitted while an application and its components are in production mode. The target audience for log and message events is users and administrators of the application and the components that make up the application. Log events are normally the only events available when a problem is first detected, and are typically used during both problem recovery and problem resolution.

Diagnostic events, which are commonly referred to as *trace events*, are used to capture internal diagnostic information about a component, and are usually not emitted or available during normal deployment and operation. The target audience for diagnostic events is the developers of the components that make up the business application. Diagnostic events are typically used when trying to resolve problems within a component, such as a software failure, but are sometimes used to diagnose other problems, especially when the information provided by the log events is not sufficient to resolve the problem. Diagnostic events are typically used when trying to resolve a problem.

A *Common Base Event* is a common structure for an event. It defines common fields, the values that these fields can take, and the exact meanings of these values for an event. Common Base Events are primarily used to represent log events.

The structure of the Common Base Event

A *Common Base Event* is a common structure for an event. It defines common fields, the values that these fields can take, and the exact meanings of these values for an event.

The Common Base Event contains several structural elements. These elements include:

- Common header information
- Component identification, both source and reporter
- Situation information
- Message data
- Extended data
- Context data
- Associated events and association engine

Each of these structural elements has its own embedded elements and attributes.

The following table presents a summary of all the fields in the Common Base Event and their usage requirements for problem determination events. This table shows whether a particular element or attribute is required, recommended, optional, prohibited, or discouraged for log events, and the base specification.

Field name	Log events	Base specification
Version	Required	Required
creationTime	Required	Required
severity	Required	Optional
Msg	Required	Optional
sourceComponentId*	Required	Required
sourceComponentId.location	Required	Required
sourceComponentId.locationType	Required	Required

sourceComponentId.component	Required	Required
sourceComponentId.subComponent	Required	Required
sourceComponentId.componentIdType	Required	Required
sourceComponentId.componentType	Required	Required
sourceComponentId.application	Recommended	Optional
sourceComponentId.instanceId	Recommended	Optional
sourceComponentId.processId	Recommended	Optional
sourceComponentId.threadId	Recommended	Optional
sourceComponentId.executionEnvironment	Optional	Optional
situation*	Required	Required
situation.categoryName	Required	Required
situation.situationType*	Required	Required
situation.situationType.reasoningScope	Required	Required
situation.situationType.(specific Situation Type elements)	Required	Required
msgDataElement*	Recommended	Optional
msgDataElement .msgId	Recommended	Optional
msgDataElement .msgIdType	Recommended	Optional
msgDataElement .msgCatalogId	Recommended	Optional
msgDataElement .msgCatalogTokens	Recommended	Optional
msgDataElement .msgCatalog	Recommended	Optional
msgDataElement .msgCatalogType	Recommended	Optional
msgDataElement .msgLocale	Recommended	Optional
extensionName	Recommended	Optional
localInstanceId	Optional	Optional
globalInstanceId	Optional	Optional
priority	Discouraged	Optional
repeatCount	Optional	Optional
elapsedTime	Optional	Optional
sequenceNumber	Optional	Optional
reporterComponentId*	Optional	Optional
reporterComponentId.location	Required (2)	Required (2)
reporterComponentId.locationType	Required (2)	Required (2)
reporterComponentId.component	Required (2)	Required (2)
reporterComponentId.subComponent	Required (2)	Required (2)
reporterComponentId.componentIdType	Required (2)	Required (2)
reporterComponentId.componentType	Required (2)	Required (2)
reporterComponentId.instanceId	Optional	Optional
reporterComponentId.processId	Optional	Optional
reporterComponentId.threadId	Optional	Optional
reporterComponentId.application	Optional	Optional
reporterComponentId.executionEnvironment	Optional	Optional
extendedDataElements*	Note 3	Optional

contextDataElements*	Note 4	Optional
associatedEvents*	Note 5	Optional

Notes:

- Items followed by an asterisk (*) are elements that consist of sub elements and attributes. The fields in those elements are listed in the table directly following the parent element name.
- Some of the elements are optional, but when included, they include sub elements and attributes that are required. For example, the reporterComponentId element has a ComponentIdentification type. The component attribute in ComponentIdentification is required. Therefore, the reporterComponentId.component attribute is required, but only when the reporterComponentId parent element is included.
- The extendedDataElements element can be included multiple times to supply extended data information. See the Extended data section for more information on required and recommended extended data element values.
- The contextDataElements element can be included multiple times to supply context data information.
- The associatedEvents element can be included multiple times to supply correlation data. No recommended uses of this element exist for the producers of problem determination data, and the use of this element is discouraged.

Common header information

This topic provides additional information about how to format and use these fields for problem determination events, which can be used to clarify and extend the information provided in the other documents.

The Common Base Event specification [CBE101] provides information on the required format of these fields and the Common Base Event Developer's Guide [CBEBASE] provides general usage guidelines.

The common header information in the Common Base Event includes the following information about an event:

- Version: The version of this Common Base Event
- creationTime: The date and time when the event generated
- Severity and priority: The severity of the condition (situation) that is identified by the event
- extensionName: The type of event that was captured
- localInstanceId and globalInstanceId: Identifiers that can be used to quickly identify a specific event within a set of events
- repeatCount and elapsedTime: Information that supports a system to efficiently report multiple events of the same type, by consolidating those events into a single event
- sequenceNumber: Sequence information that supports a system to order a set of events in other ways than time of capture

severity

All problem determination events must provide an indication as to the relative severity of the condition (situation) being reported by providing appropriate values for the severity field in the Common Base Event. The severity field is required for problem determination events. This field is more restrictive than the base specification for the Common Base Event, which lists this field as optional because effective and efficient problem determination requires the ability to quickly identify the information that is needed to resolve a problem as well as prioritize the problems that need addressing. Typically, the following values are used for problem determination events:

10	Information	Log information events, normal conditions, and events that are supplied to clarify operations, for example, state transitions, operational changes. These events typically do not require administrator action or intervention.
20	Harmless	Similar to information events, but are used to capture audit items, such as state transitions or operational changes. These events typically do not require administrator action or intervention.
30	Warning	Warnings typically represent recoverable errors, for example a failure that the system can correct. These events can require administrator action or intervention.
40	Minor	Minor errors describe events that represent an unrecoverable error within a component. The failure affects the component ability to service some requests. The business application can continue to perform its normal functions, but its overall operation might be degraded. These events require administrator action or intervention to address the condition.
50	Critical	Critical errors describe events that represent an unrecoverable error within a component. The failure significantly affects the component ability to service most requests. The business application can continue most, but not all of its normal functions and its overall operation might be degraded. These events require administrator action or intervention to address the condition.
60	Fatal	Fatal errors describe events that represent an unrecoverable error within a component. The failure usually results in the complete failure of the component. The business application can continue some normal functions, but its overall operation might be degraded. These events require administrator action or intervention to address the condition.

msg

Refer to “Message data” on page 73 for information on this attribute.

priority

The use of the priority field is discouraged for problem determination events. The severity field is typically used to communicate and evaluate the importance of problem determination events. Use the priority field to enhance the information that is provided in the severity field, that is. prioritize events of the same severity.

extensionName
The extensionName field is used to communicate the type of event that is reported, for example, what general class of events is being reported. In many cases this field provides an indication of what additional data you can expect with the event, for example, optional data values.

repeatCount
The repeatCount field is valid for problem determination events, but is not typically used or supplied by the event producers. This field is used for data reduction and consolidation by event management and analysis systems.

elapsedTime
The elapsedTime field is valid for problem determination events, but is not typically used or supplied by the event producers. This field is used for data reduction and consolidation by event management and analysis systems.

sequenceNumber
The sequenceNumber field is valid for problem determination events. It is typically used only by event producers when the granularity of the event time stamp (the creationTime field) is not sufficient in ordering events. The sequenceNumber field is typically used to sequence events that have the same time stamp value.

Event management and analysis systems can use the sequenceNumber field for a number of reasons, including providing alternative sequencing, not necessarily based on a time stamp. The recommendations here are provided primarily for event producers.

Component identification for source and reporter

The component identification fields in the Common Base Event are used to indicate which component in the system is experiencing the condition that is described by the event (the sourceComponentID) and which component emitted the event (the reporterComponentID). Typically, these components are the same, in which case only the sourceComponentID is supplied. Some notes and scenarios on when to use these two elements in the Common Base Event:

- The sourceComponentID is always used to identify the component experiencing the condition that is described by the event.
- The reporterComponentID is used to identify the component that actually produced and emitted the event. This element is typically used only within events that are emitted by a component that is monitoring another component and providing operational information regarding that component. The monitoring component (for example, a Tivoli agent or hardware device driver) is identified by the reporterComponentID and the component being monitored (for example, a monitored server or hardware device) is identified by the sourceComponentID.

  A potential misuse of the reporterComponentID is to identify a component that provides event conversion or management services for a component, for example, identifying an adapter that transforms the events that are captured by a component into Common Base Event format. The event conversion function is considered an extension of the component and not identified separately.

The information that is used to identify a component in the system is the same, regardless of whether it is the source component or reporter component:

location locationType	Component location	Identifies the location of the component.
component componentType	Component name	Identifies the asset name of the component, as well as the type of component.
subcomponent	Subcomponent name	Identifies a specific part or subcomponent of a component, for example a software module or hardware part.

application	Business application name	Identifies the business application or process the component is a part of and provides services for.
instanceId	Operational instance	Identifies the operational instance of a component, that is the actual running instance of the component.
processId threadId	Operational instance	Identifies the operational instance of a component within the context of a software operating system, that is he operating system process and thread running when the event was produced.
executionEnvironment	Operational instance Component location	Provides additional information about the operational instance of a component or its location by identifying the name of the environment hosting the operational instance of the component, for example the operating system name for a software application, the application server name for a Java 2 Platform, Enterprise Edition (J2EE) application, or the hardware server type for a hardware part.

The Common Base Event specification [CBE101] provides information on the required format of these fields and the Common Base Event Developer's Guide [CBEBASE] provides general usage guidelines. This section provides additional information about how to format and use some of these fields for problem determination events, which can be used to clarify and extend the information that is provided in the other documents.

Component

The Component field in a problem determination event is used to identify the manageable asset that is associated with the event. A manageable asset is open for interpretation, but a good working definition is a manageable asset represents a hardware or software component that can be separately obtained or developed, deployed, managed, and serviced. Examples of typical component names are:

- IBM eServer xSeries model x330
- IBM WebSphere Application Server version 5.1 (5.1 is the version number)
- Microsoft Windows 2000
- The name of an internally developed software application for a component

subComponent

The Subcomponent field in a problem determination event identifies the specific part of a component that is associated with the event. The subcomponent name is typically not a manageable asset, but provides internal diagnostic information when diagnosing an internal defect within a component, that is What part failed? Examples of typical subcomponents and their names are:

- Intel Pentium processor within a server system (Intel Pentium IV Processor)
- the enterprise bean container within a Web application server (enterprise bean container)
- the task manager within an operating system (Linux Kernel Task Manager)
- the name of a Java class and method (myclass.mycompany.com or myclass.mycompany.com.methodname).

The format of a subcomponent name is determined by the component, but use the convention shown previously for naming a Java class or the combination of a Java class and method is followed. The subcomponent field is required in the Common Base Event.

componentIdType

The componentIdType field is required by the Common Base Event specification, but provides minimal value for problem determination events. For problem determination events, the use of the application value is discouraged. The componentIdType field identifies the type of component; the application is identified by the application field.

application

The application field is listed as an optional value within the Common Base Event specification, but provide it within problem determination events whenever it this value is available. The only reason this field is not required for problem determination events is that instances exist where the issuing component might not be aware of the overall business application.

instanceId

The instanceId field is listed as an optional value within the Common Base Event specification, but provide this value within problem determination events whenever it is available.

Always provide the instanceID when a software component is identified and identify the operational instance of the component (for example, which operation instance of an installed software image is actually associated with the event). Provide this value for hardware components when these components support the concept of operational instances.

The format of the supplied value is defined by the component, but must be a value that an analysis system can use (either human or programmatic) to identify the specific running instance of the identified component. Examples include:

- **cell, node, server** name for the IBM WebSphere Application Server
- **deployed EAR file name** for a Java enterprise bean
- **serial number** for a hardware processor

processId

The processId field is listed as an optional value within the Common Base Event specification, but provide this value for problem determination events whenever it is available and applicable. Always provide this value for software-generated events, and identify the operating system process that is associated with the component that is identified in the event. Match the format of the thread ID with the format of the operating system (or other running environment, such as a Java virtual machine). This field is typically not applicable or used for events that are emitted by hardware (for example, firmware).

threadId

The threadId field is listed as an optional value within the Common Base Event specification, but provide this value for problem determination events whenever it is available and applicable. Always provide for software-generated events, and identify the active operating system thread when the event was detected or issued. A notable exception to this recommendation is some operating systems or running environments do not support threads. Match the format of the thread ID with the format of the operating system (or other running environment, such as a Java virtual machine). This field is typically not applicable or used for events that are emitted by hardware (for example, firmware).

executionEnvironment

The executionEnvironment field, when used, identifies the immediate running environment that is used by the component being identified. Some examples are:

- the operating system name when the component is a native software application.
- the operating system/Java virtual machine name when the component is a Java 2 Platform, Standard Edition (J2SE) application.
- the Web server name when the component is a servlet.
- the portal server name when the component is a portlet.
- the application server name when the component is an enterprise bean.

The Common Base Event specification [CBE101] provides information on the required format of these fields and the Common Base Event Developer's Guide [CBEBASE] provides general usage guidelines.

Situation information

The situation information is used to classify the condition that is reported by an event into a common set of situations.

The Common Base Event specification [CBE101] provides information on the set of situations defined for the Common Base Event, with the values and formats that are used to describe these situations. The Common Base Event Developer's Guide [CBEBASE] provides general usage guidelines.

Consider the following points regarding situation information for problem determination events:

- Whenever possible, use the situation categorizations and qualifiers that are described in the base Common Base Event specification. Avoid using your own situation definitions as much as possible.
- Not all messages and logs can be classified using the situation definitions that are supplied in the base Common Base Event specification. You can use the OtherSituation categorization to provide your own situation information, but the recommended course of action for problem determination events is to use the ReportSituation categorization, with reportCategory=Log.
- Warning events can be confusing. A warning event (that is an event with severity=warning) typically indicates a recoverable failure, but the situation settings can be interpreted as unrecoverable failures (for example ConnectSituation, successDisposition=UNSUCCESSFUL). Use the appropriate situation categorization so the severity setting indicates the severity of the situation, that is whether the component recovered from the failure.
- The recommended setting for the reasoningScope value is EXTERNAL for all message events.

Message data

All problem determination Common Base Events must provide human readable text that describes the specific reported event within the msg field of the Common Base Event.

The text that is associated with events representing actual messages or log entries is expected to be translated and localized. Include the msgDataElement element in the Common Base Event whenever internationalized text is provided in the event. This element provides information about how the message text is created and how to interpret it. This information is particularly invaluable when trying to interpret the event programmatically or when trying to interpret the message independent of the locale or language that is used to format the message text.

Prerequisite: Understand the concepts that are associated with creating internationalized messages. A good source of education on these concepts is provided by the documentation that is associated with internationalization of Java information and the usage of resource bundles within the Java language.

The msgDataElement element in the Common Base Event includes the following information about the value of the msg field that is provided with an event:

- The locale of the supplied message text, which identifies how the locale-independent fields within the message are formatted, as well as the language of the message (msgLocale).
- A locale-independent identifier that is associated with the message that can be used to interpret the message independent of the message language, message locale, and message format (msgId and msgIdType).
- Information on how a translated message is created, including:
  - The identifier that is used to retrieve the message template (msgCatalogId).
  - The name and type of message catalog that are used to retrieve the message template (msgCatalog and msgCatalogType).
  - Any locale-independent information that is inserted into the message template to create the final message (msgCatalogTokens).

The Common Base Event specification [CBE101] provides information on the required format of these fields and the Common Base Event Developer's Guide [CBEBASE] provides general usage guidelines. This section provides additional information about how to format and use these fields for problem determination events.

msg

All message, log, and trace events must provide a human-readable message in the msg field of the Common Base Event. The msg field is required for problem determination events, both log events and diagnostic events. This field is more restrictive than the base specification for the Common Base Event, which lists this field as optional; effective and efficient problem determination requires the ability to quickly identify the reported condition. The format and usage of this message is component-specific, but use the following general guidelines:

- Expect the message text that is supplied with messages and log events to be internationalized.
- Provide the locale of the supplied message text with the msgLocale field in the msgDataElement element of the Common Base Event.
- Provide additional information regarding the format and construction of internationalized messages whenever possible, using the msgDataElement element of the Common Base Event.

msgLocale

Provide the message locale whenever message text is provided within the Common Base Event, as is the case with all problem determination events. The msgLocale field is listed as an optional value within the Common Base Event specification, but provide this information within problem determination events whenever possible. The reason this field is not required for problem determination events is that instances exist where the locale information is not provided or available when formatting the Common Base Event.

msgId and msgIdType

Several companies include a locale-independent identifier within internationalized message text that you can use to interpret the described condition by the message text, independent of the message. For example, most messages issued by IBM software look like `IEE890I WTO Buffers in console backup storage = 1024`, where a unique, locale-independent identifier `IEE890I` precedes the translated message text. This identifier provides a way to uniquely detect and identify a message independent of location and language. This detection is invaluable for locale-independent and programmatic analysis.

The msgId field is listed as an optional value within the Common Base Event specification, but it must be provided within problem determination events whenever this identifier is included in the message text. Likewise, the msgIdType field is listed as an optional value within the Common Base Event specification, but it must be provided within problem determination events whenever a value is supplied for msgId. Do not supply these fields when the message text is not translated or localized, for example, for trace events.

msgCatalogId

The msgCatalogId field is listed as an optional value within the Common Base Event specification, but provide this value whenever the Common Base Event includes localized or translated message text, for example when providing problem determination events that represent issued messages or log events. This field is not required for problem determination events because not all problem determination events include translated message text Some cases exist where the value is not provided or available when formatting the Common Base Event. Do not supply this field when the message text is not translated or localized, for example, for trace events.

msgCatalogTokens

The msgCatalogTokens field is listed as an optional value within the Common Base Event specification, but provide this value whenever the Common Base Event includes localized or translated message text, for example when providing problem determination events that represent issued messages or log events. This field is not required for problem determination events because not all problem determination events include translated message text, and cases exist where the value is not

provided or available when formatting the Common Base Event. This value contains the list of locale-independent values or message tokens that are inserted into the localized message text when creating a translated message.

These values are difficult to extract from a translated message without knowing the translated message template that is used to create the message. Do not supply this field when the message text is not translated or localized

The Common Base Event provides several mechanisms for providing additional data about an event, including this field, extended data elements, and extensions to the schema. Always use the msgCatalogTokens field to supply the list of message tokens that is included in the message text associated with an event. These values can also be supplied in other parts of the Common Base Event, but they must be included in this field.

msgCatalog and msgCatalogType
The msgCatalog and msgCatalogType fields are listed as optional values within the Common Base Event specification, but provide this value whenever the Common Base Event includes localized or translated message text, for example when providing problem determination events that represent issued messages or log events. These fields are not required for problem determination events because not all problem determination events include translated message text, and cases exist where the values are not provided or available when formatting the Common Base Event. Do not complete these fields when the message text has is not translated or localized, for example, for trace events.

Extended data

The Common Base Event provides several methods for including this additional data, including extending the Common Base Event schema or supplying one or more ExtendedDataElement elements within the Common Base Event, which is the preferred approach.

The base information that is included in a Common Base Event might not be sufficient to represent all of the information captured by a component when creating a problem determination event.

Use an ExtendedDataElement element to represent a single data item. A Common Base Event can contain more than one of these elements, essentially one for each additional data item. A hint to the number and type of ExtendedDataElement elements is supplied by the extensionName value, but this information is only a hint. The usage of the attributes in the ExtendedDataElement element for problem determination events is the same as those for any other Common Base Event.

Sample Common Base Event instance

This XML document is an example of a Common Base Event instance that is generated by a WebSphere Application Server application.

Use the following example for reference:

```
<CommonBaseEvent  creationTime="2004-09-18T04:03:28.484Z"
   globalInstanceId="myhost:1095479647062:1899"
   msg="WSVR0024I: Server server1 stopped"
   severity="10"
   version="1.0.1">

 ... several extendedDataElements for WebSphere Application Server internal use only ...

<sourceComponentId  component="com.ibm.ws.runtime.component.ServerCollaborator"
    componentIdType="Unknown"
    executionEnvironment="Windows 2000[x86]#5.0"
    instanceId="myhost\myhost\server1"
    location="myhost"
    locationType="Hostname"
    processId="1095479647062"
    subComponent="Unknown"
    threadId="Alarm : 0"
    componentType="http://www.ibm.com/namespaces/autonomic/WebSphereApplicationServer"/>
```

```
 <msgDataElement msgLocale="en_US">
  <msgCatalogTokens value="server1"/>
  <msgId>WSVR0024I< /msgId>
  <msgCatalogId>WSVR0024I< /msgCatalogId>
  <msgCatalog>com.ibm.ws.runtime.runtime< /msgCatalog>
 </msgDataElement>

 <situation categoryName="ReportSituation">
  <situationType xsi:type="ReportSituation" reasoningScope="EXTERNAL" reportCategory="LOG"/>
 </situation>

</CommonBaseEvent>
```

A number of extendedDataElement elements in the XML are used by WebSphere Application Server, but are not for application use because these elements might change.

The CommonBaseEvent element defines the Common Base Event instance. This element has a set of attributes that are common for all Common Base Events. This set includes the extensionName attribute, which defines the type or class of the Common Base Event instance, the creation time, severity, and priority.

Nested within the CommonBaseEvent element are elements giving more detail about the situation. The first of these elements is the situation element. This classification is standardized.

The CommonBaseEvent element also includes the sourceComponentId and the (optional) reporterComponentId elements. The sourceComponentId element describes where the situation occurred; the reporterComponentId describes where the situation is detected. If the sourceComponentId and the reporterComponentId elements are the same, the reporterComponentId element is omitted.

The attributes of both the sourceComponentId and the reporterComponentId elements are the same. They identify the component type, name, operating system, and network location. The content of these attributes provides vertical correlation of the stack of IT resources that are active when the Common Base Event is created.

Also included in the CommonBaseEvent element are contextDataElements elements that describe the context in which the situation occurred. This context correlates Common Base Event instances that are part of the same work. This correlation is called *horizontal correlation* because an instance of a particular context type correlates events at the same level of abstraction, for example at the business level, the application level, or at the middleware level.

FExtended data elements contain additional data that is used to describe a situation. In this example, an extended data element is added by WebSphere Application Server to describe the Java 2 Platform, Enterprise Edition (J2EE) component that generated the Common Base Event instance and some application data.

Sample Common Base Event template

The content handler uses template information to fill in blanks in the Common Base Event when the Common Base Event complete method is called.

Components that use the WebSphere Application Server event factory home can include a Common Base Event template XML file to provide data to populate Common Base Events. Information that is already supplied in the event is not overridden if the same field is supplied in the template.

The following example illustrates a Common Base Event template:

```
<?xml version="1.0" encoding="UTF-8"?>

<TemplateEvent
```

```
  version="1.0.1"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="templateEvent.xsd">

 <CommonBaseEvent
  <sourceComponentId application="My Application" component="com.ibm.componentX"/>
  <extendedDataElements name="Sample ExtendedDataElement name" type="string">
   <values>Sample ExtendedDataElement value</values>
  </extendedDataElements>
 </CommonBaseEvent>

</TemplateEvent>
```

Component identification for problem determination

This topic describes types of problem determination events.

A business application is made up of multiple components. A component can be made up of several internal subcomponents. Consistent application of these concepts is critical for effective problem determination of a business application; all of the parts of the application must use the same concepts and assumptions when creating and formatting events. Use the following definitions and examples when creating Common Base Events for problem determination.

Business application
A business application is the business logic and business data that is used to address a set of specific business requirements. A business application consists of several components of multiple types, combined in a unique manner by an enterprise, to provide the functions and resources that are needed to address those requirements. The primary creator and manager of a business application is the enterprise, and each enterprise or company creates unique business applications. Examples of business applications are the Payroll Application for the ACME Corporation and the Inventory Application for Spacely Sprockets.

Components
A business application is created and managed by the enterprise as a set of components. Components are deployable assets, which are developed either by the enterprise or a vendor, and managed by the enterprise. A component might be created by the enterprise, typically for use within a specific business application. For example, the ACME Corporation might create a set of enterprise beans to represent the business logic that is required by their Payroll Application. A component might also be an asset that is produced by a vendor and acquired by an enterprise. Examples of these components are hardware products, such as IBM eServers or Sun Solaris systems, or software products, such as IBM WebSphere Application Server, Oracle Database Servers.

Subcomponents
A specific component, depending on its complexity, might consist of several subcomponents. For example, the IBM WebSphere Application Server consists of many subcomponents, such as the enterprise bean container and the servlet engine. Subcomponent information is typically used only by the creator of the component to service the component, and as such are not separately deployable or manageable resources in the enterprise. The enterprise might deploy a change or update to a subcomponent, but only upon guidance from the component vendor and as part of the vendor's component. For example, a software fix for the enterprise bean container of the IBM WebSphere Application Server is packaged and deployed as a software update to the IBM WebSphere Application Server. Replacement of the processor in an IBM eServer is deployed as a physical part, but only as a part of the original deployed component, the IBM eServer.

Logging Common Base Events in WebSphere Application Server

This topic describes how WebSphere Application Server takes advantage of the Common Base Events.

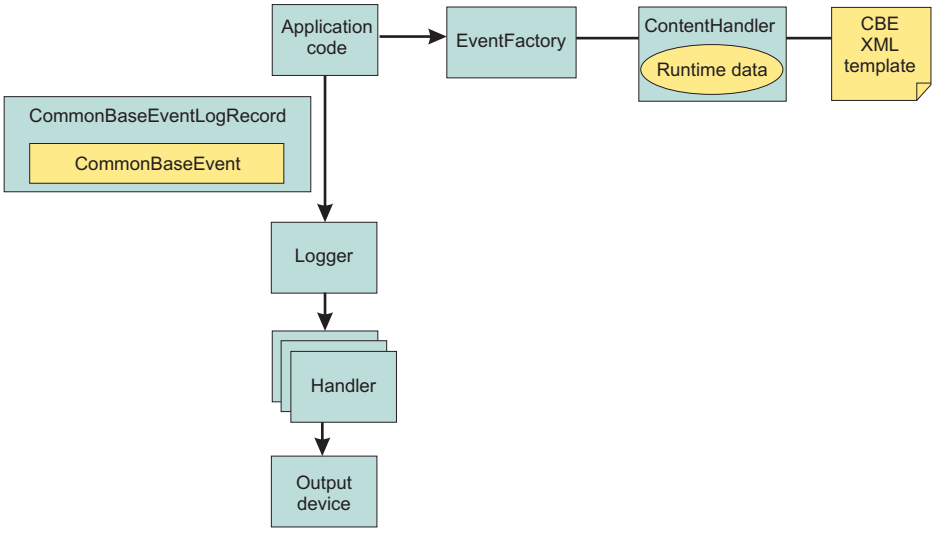
WebSphere Application Server uses Common Base Events within its logging framework. Common Base Events can be created explicitly and then logged through the Java logging API, or can be created implicitly

by using the Java logging API directly. For Common Base Event creation, the application server environment provides a Common Base Event factory with a content handler that provides both runtime data and template data for Common Base Events.

Logging with Common Base Event API and the Java logging API

In cases where the events that are generated by the Java logging API are insufficient to describe the event that needs capturing, you can create Common Base Events with the Common Base Event factory APIs.

When you create a Common Base Event, you can add data to the Common Base Event before it is logged. The following diagram illustrates how application code can create and log Common Base Events:



WebSphere Application Server is configured to use an event factory that automatically populates WebSphere Application Server-specific information into the Common Base Events that it generates. In general, it is good practice to create events using the WebSphere Application Server default Common Base Event factory because this approach ensures consistency of Common Base Event content across events. However, you can create and use other Common Base Event factories.

- Application code invokes the createCommonBaseEvent method on the EventFactory class to create a CommonBaseEvent.
- Application code wraps CommonBaseEvent event in a CommonBaseEventLogRecord record, and adds event-specific data.
- Application code calls the CommonBaseEvent event complete method.
- The CommonBaseEvent event invokes the ContentHandler completeEvent method.
- The ContentHandler handler adds XML template data to the CommonBaseEvent event. Not all ContentHandler handlers support templates.
- The ContentHandler handler adds runtime data to the CommonBaseEvent event.
- Application code passes the CommonBaseEventLogRecord record to the logger using the Logger.log method.
- Logger passes CommonBaseEventLogRecord record to Handlers.
- Handlers format data and write to the output device.

After completing all the above steps you will have a Common Base event based on your configuration settings.

Common Base Event content handler:

Content handlers populate data into Common Base Events when the Common Base Event complete method is invoked. You can associate content handlers with Common Base Event templates, which provide default information to transfer into each Common Base Event. Content handlers might also provide any other information that is relevant to completing the population of the Common Base Event, such as appropriate runtime defaults.

The use of content handlers ensures consistency of field use in the Common Base Event within a component or within a set of components that share the same runtime. For example, some content handlers support the specification of a template. If used consistently across a component, this template ensures that all events for that component have the same template information filled in. Similarly, some content handlers can also supply runtime information to their associated Common Base Events. If consistently used throughout the entire runtime, runtime information ensures that all events use runtime data in a similar way.

The event factory home that is used in the WebSphere Application Server runtime is associated with a content handler that both reads from a template, and supplies runtime data. Have components use Event Factories that are obtained from this event factory home with their own templates, to produce consistency between application events and server events.

More details can be found in“Creating custom Common Base Event content handlers” or the API documentation for org.eclipse.hyades.logging.events.cbe.ContentHandler at www.eclipse.org/hyades.

Creating custom Common Base Event content handlers:

Create a custom Common Base Event content handler or template to automate configuration or values for specific events.

A *content handler* is an object that automatically sets the property values of each event based on any arbitrary policies that you want to use.

The following content handler classes were added to WebSphere Application Server to facilitate the use of the Common Base Event infrastructure:

Class Name	Description
WsContentHandlerImpl	This provides an implementation of org.eclipse.hyades.logging.events.cbe.ContentHandler specifically for use in the WebSphere Application Server environment. This content handler completes Common Base Events using information from the WebSphere Application Server runtime, and it uses the same content handler as is used internally by the WebSphere Application Server when completing Common Base Events for logging.
WsTemplateContentHandlerImpl	This provides the same function as WsContentHandlerImpl, but it extends the org.eclipse.hyades.logging.events.cbe.impl.TemplateContentHandlerImpl class to enable the use of a Common Base Event template. Template content takes precedence in cases where the template data specifies values for the same Common Base Event fields as does the WsContentHandlerImpl.

In some situations, you might want some event property data set automatically for every event that you create. This automation is a way to fill in certain standard values that do not change, such as the application name, or to set some properties based on information that is available from the runtime environment, like creation time or thread information. You can set property data automatically by creating a content handler.

- Use the following code sample to implement the CustomContentHandler class:

```
public class CustomContentHandler extends WsContentHandlerImpl {

 public CustomContentHandler() {
  super();
  // TODO Custom initialization code goes here
 }

 public void completeEvent(CommonBaseEvent cbe) throws CompletionException {
  // following code will add WAS content to the Content Base Event
  super.completeEvent(cbe);
  // TODO Custom content can be added to the Content Base Event here
 }
}
```

- The following shows how to implement the CustomTemplateContentHandler class:

```
public class CustomTemplateContentHandler extends WsTemplateContentHandlerImpl {

 public CustomTemplateContentHandler() {
  super();
  // TODO Custom initialization code goes here
 }

 public void completeEvent(CommonBaseEvent cbe) throws CompletionException {
  // following code will add WAS content to the Content Base Event
  super.completeEvent(cbe);
  // TODO Custom content can be added to the Content Base Event here
 }
}
```

You now have a content handler or a custom content handler template based on the settings that you specified.

Common Base Event factory home:

Event Factory homes provide Event Factory instantiation that is based on a unique factory name.

Event Factory home implementations are tightly coupled with content handlers that are used to populate Common Base Events with template or default data. Event Factory instances are maintained by the associated Event Factory home, based on their unique name. For example, when application code requests a named Event Factory, the newly created Event Factory instance is returned and persisted for future requests for that named Event Factory. An abstract Event Factory home class provides the implementation for the APIs in the Event Factory home interface. Implementers extend the abstract Event Factory home class and implement the createContentHandler API to create a typed content handler that is based on the type of Event Factory home implementation.

In WebSphere Application Server, the default Event Factory home that is obtained with a call to EventFactoryContext.getInstance.getEventFactoryHome method is associated with a ContentHandler handler capable of supplying both event template information, as well as WebSphere Application Server runtime default information.

More details can be found in the API documentation for org.eclipse.hyades.logging.events.cbe.EventFactoryHome at www.eclipse.org/hyades.

Creating custom Common Base Event factory homes:

Use custom Common Base Event factory homes to control configuration and implementation of unique Event Factories.

Event Factory Homes create and provide homes for Event Factory instances. Each Event Factory Home has a Content Handler. This Content Handler is assigned to every Event Factory the Event Factory Home creates. In turn, when a Common Base Event is created, the Content Handler from the Event Factory is

assigned to it. Event Factory instances are maintained by the associated Event Factory Home, based on their unique name. For example, when application code requests a named Event Factory, the newly created Event Factory instance is returned and persisted for future requests for that named Event Factory.

The following classes were added to facilitate the use of Event Factory homes for logging Common Base Events:

Class Name	Description
WsEventFactoryHomeImpl	This class extends the org.eclipse.hyades.logging.events.cbe.impl.AbstractEventFactoryHome class. This Event Factory Home returns Event Factory instances associated with the WsContentHandlerImpl Content Handler. The WsContentHandlerImpl is the Content Handler used by the WebSphere Application Server by default when no Event Factory template is in use.
WsTemplateEventFactory HomeImpl	This class extends the org.eclipse.hyades.logging.events.cbe.impl.EventXMLFileEventFactoryHomeImpl class. This Event Factory Home returns Event Factory instances associated with the WsTemplateContentHandlerImpl Content Handler. The WsTemplateContentHandlerImpl is the Content Handler used by the WebSphere Application Server when an Event Factory template is required.

Custom event factory homes support the use of Common Base Event for logging in WebSphere Application Server and make logging easy and consistent between the WebSphere Application Server runtime and the exploiters of this API. The CustomEventFactoryHome and CustomTemplateEventFactoryHome classes will be used to obtain an event factory. These classes are there to make sure the correct content handler is being used with a particular event factory. The CustomEventFactoryHelper class is an example of how the infrastructure provider can hide the factory selection details from infrastructure users, using their own set of parameters to decide which the appropriate event factory is.

- The following code samples provide examples of how to implement and use the CustomEventFactoryHome class.
  1. Implementation of the CustomEventFactoryHome class is as follows:

```
public class CustomEventFactoryHome extends AbstractEventFactoryHome {

 public CustomEventFactoryHome() {
  super();
  // TODO Custom intialization code goes here
 }

 public ContentHandler createContentHandler(String arg0) {
  // Always use custom content handler
  return resolveContentHandler();
 }

 public ContentHandler resolveContentHandler() {
  // Always use custom content handler
  return new CustomContentHandler();
 }
}
```

  2. The following is an example of how to use the CustomEventFactoryHome class:

```
// get the event factory
  EventFactory eventFactory=(new CustomEventFactoryHome()).getEventFactory("XYZ");
  // create an event - call appropriate method
  eventFactory.createCommonBaseEvent();
  // log event ...
```

- For the CustomTemplateEventFactoryHome class you can use the following code for implementation and use:

1. Implement the CustomTemplateEventFactoryHome class by using this code:

```
public class CustomTemplateEventFactoryHome extends
  EventXMLFileEventFactoryHomeImpl {

 public CustomTemplateEventFactoryHome() {
  super();
  // TODO Custom intialization code goes here
 }

 public ContentHandler createContentHandler(String arg0) {
  // Always use custom content handler
  return resolveContentHandler();
 }

 public ContentHandler resolveContentHandler() {
  // Always use custom content handler
  return new CustomTemplateContentHandler();
 }
}
```

2. Use the CustomTemplateEventFactoryHome class by following this sample code:

```
// get the event factory
  EventFactory eventFactory=(new
    CustomTemplateEventFactoryHome()).getEventFactory("XYZ");
  // create an event - call appropriate method
  eventFactory.createCommonBaseEvent();
  // log event ...
```

- The CustomEventFactoryHelper class can be implemented and used by following the code below:
  1. Implement the custom CustomEventFactoryHelper class using this code:

```
public class CustomTemplateEventFactoryHome extends
  EventXMLFileEventFactoryHomeImpl {

 public CustomTemplateEventFactoryHome() {
  super();
  // TODO Custom intialization code goes here
 }

 public ContentHandler createContentHandler(String arg0) {
  // Always use custom content handler
  return resolveContentHandler();
 }

 public ContentHandler resolveContentHandler() {
  // Always use custom content handler
  return new CustomTemplateContentHandler();
 }
}
```

Figure 4 CustomTemplateEventFactoryHome class

```
public class CustomEventFactoryHelper {
 // name of the event factory to use
 public static final String FACTORY_NAME="XYZ";

 public static EventFactory getEventFactory(String param1, String param2) {
  EventFactory factory=null;
  switch (resolveFactory(param1,param2)) {
  case 1:
   factory=(new CustomEventFactoryHome()).getEventFactory(FACTORY_NAME);
   break;
  case 2:
   factory=(new
    CustomTemplateEventFactoryHome()).getEventFactory(FACTORY_NAME);
   break;

  default:
   // Add default for event factory
```

```
      break;
     }
     return factory;
    }

    private static int resolveFactory(String param1, String param2) {
     int factory=0;
     // Add code here to resolve which factory to use
     return factory;
    }
   }
```

2. To use the CustomEventFactoryHelper class, use the following code:

```
// get the event factory
  EventFactory eventFactory=
   CustomEventFactoryHelper.getEventFactory("param1","param2","param3");
  // create an event - call appropriate method
  eventFactory.createCommonBaseEvent();
  // log event ...
```

Use the information provided here to implement a custom content factory home and the associated classes based on the settings that you specify.

Common Base Event factory context:

The event factory context provides a service to look up event factory homes. Retrieve the event factory context using a call to the EventFactoryContext.getInstance method.

Using this class, you can look up the event factory homes by name, and avoid the need to include the typed home in code. The EventFactoryHome name must be located on the class path to be found. The EventFactoryContext context also stores an EventFactoryHome name as a default, which can be obtained with a call to the EventFactoryContext.getInstance.getEventFactoryHome method.

In WebSphere Application Server, the EventFactoryContext context is configured with a default EventFactoryHome name which is associated to a ContentHandler handler that is capable of supplying both event template information, as well as WebSphere Application Server runtime default information.

More details can be found in the API documentation for org.eclipse.hyades.logging.events.cbe.EventFactory at www.eclipse.org/hyades.

Common Base Event factory:

Use event factories to create Common Base Events and complete event properties with associated content handlers.

Content handlers populate data into Common Base Events when the Common Base Event invokes the complete method. All event properties set by the application code have priority over all properties that are specified by the content handler. Event factory implementations are tightly coupled with the content handler instance, which is associated with the event factory when the event factory is instantiated. Factory instances can be retrieved only from their associated event factory home. Event factory instances are retrieved and maintained based on unique names. Event factory names are hierarchical; they are represented using the standard Java dot-delimited, name-space naming conventions.

More details can be found in the API documentation for org.eclipse.hyades.logging.events.cbe.EventFactory at www.eclipse.org/hyades.

java.util.logging -- Java logging programming interface

The java.util.logging.Logger class provides a variety of methods with which data can be logged.

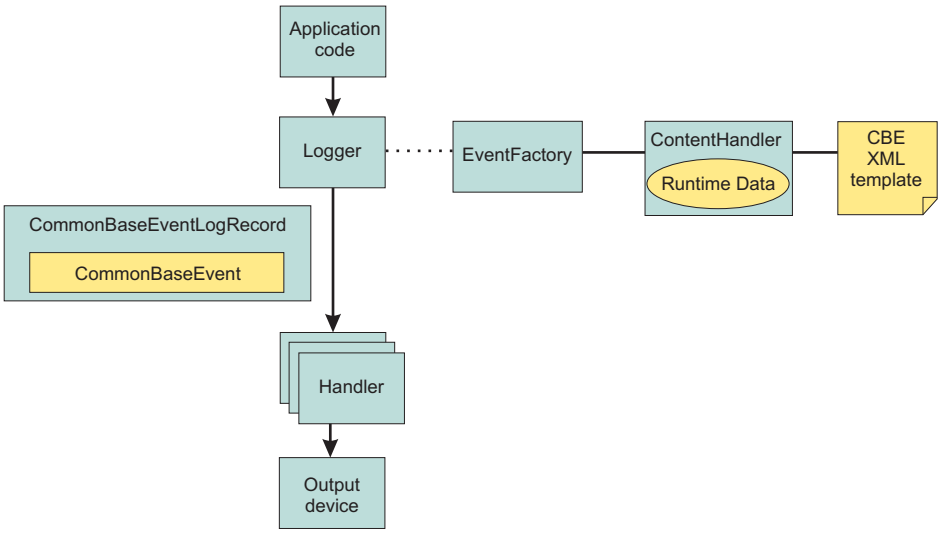
In the WebSphere Application Server, the Java logging API (java.util.logging) automatically creates Common Base Events for events that are logged at the WsLevel.DETAIL level or above (including WsLevel.DETAIL, Level.CONFIG, Level.INFO, WsLevel.AUDIT, Level.WARNING, Level.SEVERE, and WsLevel.FATAL). These Common Base Events are created using the event factory that is associated with the logger to which the message is logged. If no event factory is specified, WebSphere Application Server uses a default event factory which automatically fills in WebSphere Application Server-specific information.

The WebSphere Application Server uses a special implementation of the java.util.logging.Logger class that automatically creates Common Base Events for the following methods:

- config
- info
- warning
- severe
- log: All variants except log(LogRecord) when used with the WsLevel.DETAIL level or more severe levels
- logp: When used with the WsLevel.DETAIL level or more severe levels
- logrb: When used with the WsLevel.DETAIL level or more severe levels

The WebSphere Application Server logger implementation is used only for named loggers for example, loggers that are instantiated with calls, such as Logger.getLogger(″com.xyz.SomeLoggerName″). Loggers instantiated with calls to the Logger.getAnonymousLogger and Logger.getLogger, or Logger.global methods do not use the WebSphere Application Server implementation, and do not automatically create Common Base Events for logging requests made to them. Log records that are logged directly with the Logger.log(LogRecord) method are not automatically converted by WebSphere Application Server loggers into Common Base Events.

The following diagram illustrates how application code can log Common Base Events:



The Java logging API processing of named loggers and message-level events proceeds as follows:

1. Application code invokes the named logger (WsLevel.DETAIL or above) with event-specific data.
2. The logger creates a Common Base Event using the createCommonBaseEvent method on the event factory that is associated with the logger.
3. The logger creates a Common Base Event using the event factory associated to the logger.
4. The logger wraps the common base event in a CommonBaseEventLogRecord record, and adds event-specific data.
5. The logger calls the Common Base Event complete method.
6. The Common Base Event invokes the ContentHandler completeEvent method.

7. The content handler adds XML template data to the Common Base Event (including for example, the component name). Not all content handlers support templates.
8. The content handler adds runtime data to the Common Base Event (including for example, the current thread name).
9. The logger passes the CommonBaseEventLogRecord record to the handlers.
10. The handlers format data and write to the output device.

Logger.properties file

Use the Logger.properties file to set logger attributes for your component.

The properties file is loaded the first time the Logger.getLogger(loggername) method is called within an application. The Logger.properties file must be either on the WebSphere Application Server class path, or the context class path.

The logging subsystem uses Common Base Events to represent all the messages in the WebSphere Application Server activity.log file. You can specify your own event factory template to be used with your loggers. Use the eventfactory property in your Logger.properties file. See "Sample Common Base Event template" on page 76 for details on the Common Base Event template.

By convention, the name of the event factory template file should be the fully qualified package name of the package using the template. The name of the file must end with the `.event.xml` extension. For example, a valid event factory template file name for the com.abc.somepackage package is:

```
com.abc.somepackage.event.xml
```

When you specify the property value for the eventfactory property in the Logger.properties file, include the full path name with no leading slash relative to the root of your class path entry. Do not include the `.event.xml` extension.

For example, if the template files from the example above are located in the `com/abc/templates` directory, the valid value for the eventfactory property is:

```
com/abc/templates/com.abc.somepackage
```

Finally, if this event factory template file is used by the com.abc.somepackage.SomeClass logger, then the following entry will appear in the Logger.properties file:

```
com.abc.somepackage.SomeClass.eventfactory=com/abc/templates/com.abc.somepackage
```

Generate Common Base Event content with the default event factory

A default Common Base Event content handler populates Common Base Events with WebSphere Application Server runtime information. This content handler can also use a Common Base Event template to populate Common Base Events.

The default content handler is used when the server creates CommonBaseEventLogRecords as would be the case in the following example:

```
// Get a named logger
Logger logger = Logger.getLogger("com.ibm.someLogger");
// Log to the logger -- implicitly the default content handler
// will be associated with the CommonBaseEvent contained in the
// CommonBaseEventLogRecord. logger.warning("MSG_KEY_001");
```

To specify a Common Base Event template in the above case, a Logger.properties file would need to be provided with an eventfactory entry for *com.ibm.someLogger*. If a valid template is found on the classpath, then the Logger's event factory will use the specified template's content in addition to the WebSphere Application Server runtime information when populating Common Base Events. If the template is not found on the classpath, or is invalid, then the Logger's event factory will only use the WebSphere Application Server runtime information when populating Common Base Events.

The default content handler is also associated with the event factory home supplied in the global event factory context. This is convenient for creating Common Base Events that need to be populated with content similar to that generated from the WebSphere Application Server:

```
// Request the event factory from the global event factory home
EventFactory eventFactory = EventFactoryContext.getInstance().getEventFactoryHome().getEventFactory(templateName);

// Create a Common Base Event
CommonBaseEvent commonBaseEvent = eventFactory.createCommonBaseEvent();

// Complete the Common Base Event using content from the template (if specified above)
// and the server runtime information.
eventFactory.getContentHandler().completeEvent(commonBaseEvent);
```

In the above example, if the template referenced by *templateName* is found on the classpath, and the template is valid, then the event factory home will return an event factory which uses a content handler that combines the template's content with the WebSphere Application Server runtime information when populating Common Base Events. If the template is not found on the classpath, or is invalid, then the event factory home will return an event factory which uses a content handler that uses only the WebSphere Application Server runtime information when populating Common Base Events.

The default content handler populates Common Base Events in the server environment with the following runtime information:

CommonBaseEvent.globalInstanceId
Value: The *unique_record_id*

Set this value only if the CommonBaseEvent.globalInstanceId value is null before the completeEvent method is called.

CommonBaseEvent.msg
Value: A localized message that is based on the MsgDataElement element.

Set this value only if the CommonBaseEvent.msg message is null before the completeEvent method is called.

CommonBaseEvent.severity
Value: Set based on the value of level set on the CommonBaseEventLogRecord record, if level >= Level.SEVERE, set to `50`; if level >= Level.WARNING, set to `30`; the default is set to `10`.

Set this value only if the CommonBaseEvent.severity value is null before the completeEvent method is called.

CommonBaseEvent.ComponentIdentification.component
Value: Set based on the LoggerName value that is set on the CommonBaseEventLogRecord record.

Set this value only if the CommonBaseEvent.ComponentIdentification.component is null before the completeEvent method is called.

CommonBaseEvent.ComponentIdentification.componentIdType
Value: ″Unknown″

Set this value only if the CommonBaseEvent.ComponentIdentification.componentIdType value is null before the completeEvent method is called.

CommonBaseEvent.ComponentIdentification.executionEnvironment
Value: `OSname[OSarch]#OSversion`

Set this value only if the CommonBaseEvent.ComponentIdentification.executionEnvironment value is null before the completeEvent method is called.

CommonBaseEvent.ComponentIdentification.instanceId
Value: `cellName\nodeName\serverName`

Set this value only if the CommonBaseEvent.ComponentIdentification.instanceId value is null before the completeEvent method is called. Set only in a server environment because this value is ignored in a client application.

CommonBaseEvent.ComponentIdentification.location
Value: The host name

Set this value only if both the CommonBaseEvent.ComponentIdentification.location and the CommonBaseEvent.ComponentIdentification.locationType values are null before the completeEvent method is called.

CommonBaseEvent.ComponentIdentification.locationType
Value: The host name

Set this value only if both the CommonBaseEvent.ComponentIdentification.location and the CommonBaseEvent.ComponentIdentification.locationType values are null before the completeEvent method is called.

CommonBaseEvent.ComponentIdentification.processId
Value: An internally generated representation of the process number.

Set this value only if the CommonBaseEvent.ComponentIdentification.processId value is null before the completeEvent method is called

CommonBaseEvent.ComponentIdentification.subComponent
Value: Set based on values of the sourceClassName and the sourceMethodName names that are set on the sourceClassName.sourceMethodName name of the CommonBaseEventLogRecord record.

Set this value only if the CommonBaseEvent.ComponentIdentification.subComponent values is null before the completeEvent method is called and both the sourceClassName and the sourceMethodName names are set.

CommonBaseEvent.ComponentIdentification.threadId
Value: Set to the value of the Java Virtual Machine (JVM) thread name.

Set this value only if the CommonBaseEvent.ComponentIdentification.threadId values is null before the completeEvent value is called.

CommonBaseEvent.ComponentIdentification.componentType
Value: http://www.ibm.com/namespaces/autonomic/WebSphereApplicationServer

Set this value only if the CommonBaseEvent.ComponentIdentification.componentType values is null before the completeEvent method is called.

CommonBaseEvent.MsgDataElement.msgLocale
Value: Set based on the default locale of the JVM.

Set this value only if the CommonBaseEvent.msg value is null before the completeEvent method is called.

CommonBaseEvent.Situation.categoryName
Value: `ReportSituation`

Set this value only if the CommonBaseEvent.Situation value is null before the completeEvent method is called.

CommonBaseEvent.Situation.situationType.type
Value: `ReportSituation`

Set this value only if the CommonBaseEvent.Situation value is null before the completeEvent method is called.

CommonBaseEvent.Situation.situationType.reasoningScope
Value: EXTERNAL

Set this value only if the CommonBaseEvent.Situation value is null before the completeEvent method is called.

CommonBaseEvent.Situation.situationType.reportCategory

Value: LOG

Set this value only if the CommonBaseEvent.Situation value is null before the completeEvent method is called.

The sourceComponentIdentification value is populated if no reporterComponentIdentification ID exists when the completeEvent method is invoked on the content handler. Otherwise, the reporterComponentIdentification ID is populated instead.

Best practices for logging Common Base Events in WebSphere Application Server

The following practices ensure consistent use of Common Base Events within your components, and between your components and WebSphere Application Server components.

Follow these guidelines:

- Use a different logger for each component. Sharing loggers across components gets in the way of associating loggers with component-specific information.
- Associate loggers with event templates that specify source component identification. This association ensures that the source of all events created with the logger is properly identified.
- Use the same template for directly created Common Base Events (events created using the Common Base Event factories) and indirectly created Common Base Events (events created using the Java logging API) within the same component.
- Avoid calling the complete method on Common Base Events until you are finished adding data to the Common Base Event and are ready to log it. This approach ensures that any decisions made by the content handler based on data already in the event are made using the final data.

The following sample `Logger.properties` file entry demonstrates how to associate the com.ibm.componentX logger with the com.ibm.componentX event factory:

```
com.ibm.componentX.eventfactory=com.ibm.componentX
```

The following sample code demonstrates the use of the same event factory setting for direct (Part 1) and indirect (Part 2) Common Base Event logging:

```
<?xml version="1.0" encoding="UTF-8"?>

<TemplateEvent
  version="1.0.1"
  xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
  xsi:noNamespaceSchemaLocation="templateEvent.xsd">

 <CommonBaseEvent
   <sourceComponentId application="My application" component="com.ibm.componentX"/>
   <extendedDataElements CommonBaseEventname="Sample ExtendedDataElement name" type="string">
   <values>Sample ExtendedDataElement value</values>
  </extendedDataElements>
 < /CommonBaseEvent>

< /TemplateEvent>
```

Chapter 6. Diagnosing problems (using diagnosis tools)

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

1. For tips on how to investigate common kinds of problems based on the component that is causing the problem, see Troubleshooting by component.
2. If deploying or running an application results in exceptions such as ClassNotFoundException, use the Class Loader Viewer to diagnose problems with class loaders.
3. If you already have an error message and want to quickly look up its explanation and recommended response, look up the message by expanding the Messages section of the Information Center under **Reference** > **Troubleshooter** > **Messages**.
4. For help in knowing where to find error and warning messages, interpreting messages, and configuring log files, see Working with message logs.
5. Difficult problems can require the use of tracing, which exposes the low-level flow of control and interactions between components. For help in understanding and using traces, see Working with trace.
6. For help in adding log and trace capability to your own application, see “Log and trace with Java logging” on page 19.
7. For help in using settings or tools to help you diagnose the problem, see Working with troubleshooting tools. Some of these tools are bundled with the product, and others are freely downloadable.
8. To learn how to work with Diagnostic Providers, see Working with Diagnostic Providers..
9. To find out how to look up documented problems, common mistakes, WebSphere Application Server prerequisites, and other problem-determination information on the WebSphere Application Server public Web site, or to obtain technical support from IBM, see Obtaining help from IBM.
10. The Troubleshoot IBM Developer Kit for Java describes debugging techniques and the diagnostic tools that are available to help you solve problems with Java. It also gives guidance on how to submit problems to IBM.
11. For current information available from IBM Support on known problems and their resolution, see the IBM Support page. For last minute updates, limitations, and known problems, refer to the Release notes section.
12. IBM Support has documents that can save you time gathering information needed to resolve this problem. Before opening a PMR, see the Must gather documents page for information to gather to send to IBM Support.

Troubleshooting class loaders

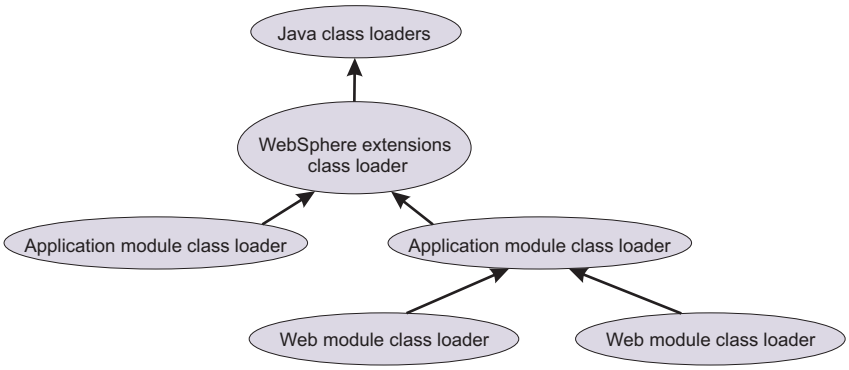
Class loaders find and load class files. For a deployed application to run properly, the class loaders that affect the application and its modules must be configured so that the application can find the files and resources that it needs. Diagnosing problems with class loaders can be complicated and time-consuming. To diagnose and fix the problems more quickly, use the administrative console Class Loader Viewer to examine class loaders and the classes loaded by each class loader.

This topic assumes that you have installed an application on a server supported by WebSphere Application Server and you want to examine class loaders used by the application or its modules. The modules can be Web modules (`.war` files) or enterprise bean (EJB) modules (`.jar` files). The Class Loader Viewer enables you to examine class loaders in a runtime environment.

This topic also assumes that you have enabled the class loader viewer service. Click **Servers > Application servers >** *server_name* **> Class Loader Viewer Service**, enable the service and restart the server.

The runtime environment of WebSphere Application Server uses the following class loaders to find and load new classes for an application in the following order:

1. The bootstrap, extensions, and CLASSPATH class loaders created by the Java virtual machine
2. A WebSphere extensions class loader
3. One or more application module class loaders that load elements of enterprise applications running in the server
4. Zero or more Web module class loaders



Each class loader is a child of the previous class loader. That is, the application module class loaders are children of the WebSphere extensions class loader, which is a child of the CLASSPATH Java class loader. Whenever a class needs to be loaded, the class loader usually delegates the request to its parent class loader. If none of the parent class loaders can find the class, the original class loader attempts to load the class. Requests can only go to a parent class loader; they cannot go to a child class loader. After a class is loaded by a class loader, any new classes that it tries to load reuse the same class loader or go up the precedence list until the class is found.

If the class loaders that load the artifacts of an application are not configured properly, the JVM might throw a class loading exception when starting or running that application. “Class loading exceptions” on page 92 describes the types of exceptions caused by improperly configured class loaders and suggests ways to use the Class Loader Viewer to correct configurations of class loaders. The types of exceptions include:

- ClassCastException
- ClassNotFoundException
- NoClassDefFoundException
- UnsatisfiedLinkError

The steps that follow describe generally how to use the Class Loader Viewer to examine class loaders and correct problems with application or class loader configurations.

- Examine a tree view that lists all installed applications and their modules. The modules can be Web modules (`.war` files) or EJB modules (`.jar` files).

  Click **Troubleshooting > Class Loader Viewer** to access the Enterprise Applications Topology page.
- Examine the class loader delegation hierarchy.

  On the Enterprise Applications Topology page, select a module to access the Class Loader Viewer page. The page lists the class loaders visible to Web and EJB modules in an installed enterprise application. This page helps you to determine which class loaders loaded files of a module and to diagnose problems with class loaders.

The delegation hierarchy is determined by the class loader delegation mode, or *class loader order*, specified for an application or Web module. The mode value can be either `Classes loaded with parent class loader first` or `Classes loaded with application class loader first`. Refer to the Configure class loaders step for more information.

- Export information on class loaders.
  1. On the Class Loader Viewer page, click **Export**.
  2. Select to open a browser or editor on the class loader information or to save the information to disk in XML format.
  3. Click **OK**, and specify any additional information requested by the system.
- Display information about class loaders visible to the module in an HTML table format.

  On the Class Loader Viewer page, click **Table View**. The Table View page displays the following information:

Class loader attribute	Description
Delegation	Indicates whether the class loader delegates the loading of the module to its parent class loader. A value of `true` implies that the class loader of the parent application is being used (`Parent first` or `Classes loaded with parent class loader first`). A value of `false` implies that the module class loader is being used (`Parent last` or `Classes loaded with application class loader first`). Refer to the Configure class loaders step for more information.
Classpath	Lists the paths over which the class loader searches for classes and resources.
Classes	Lists the names of classes loaded in the JVM by this class loader.

  The **Table View** option does not return a value when out-of-memory errors are generated. The out-of-memory errors might be related to a memory leak. To examine information about class loaders in a table, resolve the out-of-memory problem, and then click **Table View** again.
- Search class loaders.

  On the Class Loader Viewer page, click **Search** to access the Search page, on which you can search class loaders for the following:
  – Specific strings
  – Specific `.jar` files
  – The names of files in a specific directory
  – The names of files loaded by a specific class loader

  The search is case-sensitive. “Class loading exceptions” on page 92 describes several uses of the Search page.
- Configure class loaders. You can configure class loaders for the following:
  – All applications installed on a specific server.
  – A specific application
  – A specific Web module
  –

  Note: For detailed information about server, application, and Web class loaders, see the chapter on class loading in the *Developing and deploying applications* PDF book.

  Class loader configuration determines which class loader loads the classes and resource files for an application or Web module. Application and WAR module class loader configuration settings include **Class loader order** and **WAR class loader policy**.

  A **Class loader order** value can be either `Classes loaded with parent class loader first` or `Classes loaded with application class loader first`. The default is `Classes loaded with parent class loader first`. A class loader with the `Classes loaded with parent class loader first` mode delegates loading a class or resource to its immediate parent class loader before searching its classpath.

When troubleshooting class loading problems, you might need to override classes visible to a parent class loader. To override such classes with those specific to an application, set the **Class loader order** to `Classes loaded with application class loader first` on the class loader that contains the application classes on its classpath. An application can override classes visible to a parent class loader, but doing so can result in a ClassCastException or UnsatisfiedLinkError if there is a mixed use of overridden classes and non-overridden classes.

For example, under default class loader policies, a Web module has its own Web module (WAR) class loader to load its artifacts, which are typically in the `WEB-INF/classes` and `WEB-INF/lib` directories. An application module class loader is the immediate parent of this WAR class loader. To ensure that the Web module class loader searches these paths for a particular class or resource first, before delegating the load operation to the application module class loader, set the **Class loader order** of the Web module to `Classes loaded with application class loader first`.

Class loader policies determine the structure of the application and WAR module class loaders. Under the default policies, every running application EAR has its own application module class loader, and every Web module has its own WAR module class loader. The default policies ensure J2EE compliance regarding visibility and isolation among application artifacts. Changing the default policies is not suggested when troubleshooting class loading problems.

If you continue to have class loader problems, refer to "Class loading exceptions" and to the class loading chapter of the *Developing and deploying applications* PDF book.

Class loading exceptions

What kind of class-loading error do you see when you develop an application or start an installed application?

ClassCastException

A class cast exception results when the following conditions exist and can be corrected by the following actions:

- The type of the source object is not an instance of the target class (type).
- The class loader that loaded the source object (class) is different from the class loader that loaded the target class.
- The application fails to perform or improperly performs a narrow operation.

The type of the source object is not an instance of the target class (type).

This is the typical class cast exception. You can diagnose whether the source object of a cast statement is not an instance of the target class (type) by examining the class signature of the source object class, then verifying that it does not contain the target class in its ancestry and the source object class is different than the target class. You can obtain class information by inserting a simple print statement in your code. For example:

```
System.out.println( source.getClass().getName() + ":" + target.getClass().getName() );
```

Or use a `javap` command. For example:

```
javap java.util.HashMap
Compiled from "HashMap.java"
public class java.util.HashMap extends java.util.AbstractMap
             implements java.util.Map,java.lang.Cloneable,java.io.Serializable {
```

The class loader that loaded the source object (class) is different from the class loader that loaded the target class.

Assuming that the type of the source object is an instance of the target class, a class cast exception occurs when the class loader that loaded the source object's class is different that the

class loader that loaded the target class. This condition might occur when the target class is visible on the classpaths of more than one class loader in the WAS runtime environment. To correct this problem, use the Search and Search by Class Name console pages used to diagnose problems with class loaders:

1. Click **Troubleshooting > Class Loader Viewer >** *module_name* **> Search** to access the Search page.
2. For **Search String**, type the name of the class that is loaded by two class loaders.
3. For **Search Type**, select **Class/Package**.
4. Click **OK**. The Search by Class Name page is displayed, listing all class loaders that load the class.

   If there is more than one class loader listed, then the target class was loaded by more than one class loader. Because the source object is an instance of the target class, the class loader that loaded the source object class is different from the class loader that loaded the target class.
5. Return to the Class Loader Viewer page and examine the classpath to determine why two different class loaders load the class.
6. Correct your code so that the class is visible only to the appropriate class loader.

The application fails to perform or improperly performs a narrow operation.

A class cast exception can occur because, when the application is resolving a remote enterprise bean (EJB) object, the application code does not perform a narrow operation as required. The application must perform a narrow operation after looking up a remote object. Examine the application and determine whether it looks up a remote object and, if so, the result of the lookup is submitted to a narrow method.

The narrow method must be invoked according to the EJB 2.0 programming model. In particular, the target class submitted to the narrow method must be the exact, most derived interface of the EJB. This also causes a class cast exception in the WAS runtime environment. Examine the application and determine whether the target class submitted to the narrow method is a super-interface of the EJB that is specified, not the exact EJB type; if so, modify the application to invoke narrow with the exact EJB interface.

Lastly, if a class cast exception occurs during a narrow operation, verify that the narrow method is being applied to the result of a remote EJB lookup, not to a local EJB. A narrow is not used for local lookups. Examine the application or module deployment descriptor to ensure that the object being narrowed is not a local object.

ClassNotFoundException

A class not found exception results when the following conditions exist and can be corrected by the following actions:

- The class is not visible on the logical classpath of the context class loader.
- The application incorrectly uses a class loader API.
- A dependent class is not visible.

The class is not visible on the logical classpath of the context class loader.

The class not found is not in the logical class path of the class loader associated with the current thread. The logical classpath is the accumulation of all classpaths searched when a load operation is invoked on a class loader. To correct this problem, use the Search page to search by class name and by Java archive (JAR) name:

1. Click **Troubleshooting > Class Loader Viewer >** *module_name* **> Search** to access the class loader Search page.
2. For **Search String**, type the name of the class that is not found.
3. For **Search Type**, select **Class/Package**.
4. Click **OK**. The Search by Class Name page is displayed, listing all class loaders that load the class.

5. Examine the page to see if the class exists in the list.
6. If the class is not in the list, return to the Search page. For **Search String**, type the name of the `.jar` file for the class; for **Search Type**, select **JAR/Directory**.
7. Click **OK**. The Search by Path page is displayed, listing all directories that hold the JAR file.

If the JAR file is not in the list, the class likely is not in the logical class path, not readable or an alternate class is already loaded. Move the class to a location that enables it to be loaded.

The application incorrectly uses a class loader API.

An application can obtain an instance of a class loader and call either the loadClass method on that class loader, or it can call Class.forName(*class_name*, *initialize*, *class_loader*) with that class loader. The application may be incorrectly using the class loader application programming interface (API). For example, the class name is incorrect, the class is not visible on the logical classpath of that class loader, or the wrong class loader was engaged.

To correct this problem, determine whether the class exists and whether the application is properly using the class loader API. Follow the steps in The class is not visible on the logical classpath of the context class loader to determine whether the class is loaded. If the class has not been loaded, attempt to correct the application and see if the class loads. If the class is in the class path with proper permission and is not being overridden by another factory class, examine the API used to load the class.

1. Click **Troubleshooting > Class Loader Viewer >** *module_name* **> Search** to access the class loader Search page.
2. For **Search String**, type the name of the class.
3. For **Search Type**, select **Class/Package**.
4. Click **OK**. The Search by Class Name page is displayed, listing all class loaders that load the class.
5. Examine the page to see if the class exists in the list.
6. If the class is in the list and a ClassNotFound exception was thrown, then the `.jar` file or class is not in the correct context or a wrong API call in the current context was used.

   If the class is not in the list, return to the Search page and do the following:
   a. Search for the class that generated the exception; that is, the class calling Class.forName.
   b. See which class loader loads the class.
   c. Determine whether the class loader has access or can load the class not found by evaluating the class path of the class loader.

A dependent class is not visible.

When a class loader *clsldr* loads a class *cls*, the Java virtual machine (JVM) invokes *clsldr* to load the classes on which *cls* depends. Dependent classes must be visible on the logical classpath of *clsldr*, otherwise an exception occurs. This condition typically occurs when users make WebSphere Application Server classes visible to JVM, or make application classes visible to the JVM or to the WebSphere extensions class loader. For example:

- Class A depends on Class B.
- Class A is visible to the WebSphere extensions class loader.
- Class B is visible on the local classpath of a WAR module class loader, not the WebSphere extensions class loader classpath.

When the JVM loads class A using the WebSphere extensions class loader, it then attempts to load Class B using the same class loader and ultimately creates a class not found exception.

To correct this problem:

1. Make the application-specific classes visible to the appropriate application class loader.
2. Search for the class not found (Class B).
3. If Class B is in the proper location, search for the class that loads the dependent class (Class A) in the Class Load Viewer.
4. If the class is loaded and a ClassNotFound exception was thrown, then the `.jar` file or class is not in proper context or the wrong API call in the current context was used.

If no class was found, do the following:
a. Search for the class that generated the exception; that is, the class calling Class.forName.
b. See which class loader loads the class.
c. Determine whether the class loader has access or can load the class not found by evaluating the class path of the class loader.

5. Ensure that the caller class (Class B) is visible to the JVM or WebSphere extensions class loader.

NoClassDefFoundException

A no class definition found exception results when the following conditions exist and can be corrected by the following actions:

The class is not in the logical class path.
Refer to "ClassNotFoundException" on page 93 for information.

The class cannot load.
There are various reasons for a class not loading. The reasons include: failure to load the dependent class, the dependent class has a bad format, or the version number of a class.

UnsatisfiedLinkError

A linkage error results when the following conditions exist and can be corrected by the following actions:
- A user action caused the error.
- System.mapLibraryName returns the wrong library file.
- The native library is already loaded.
- A dependent native library was used.

A user action caused the error.

Several user actions can result in a linkage error:

A library extension name is incorrect for the platform.

System.loadLibrary is passed an incorrect parameter.

The library is not visible.
As a best practice, use the JVM class loader to find or load native libraries. WebSphere Application Server prints the Java library path (`java.library.path`) when starting up. If the JVM class loader is intended to load the library, verify that the path containing the native library file is in the Java library path. If not, append the path to the platform-specific native library environment variable or to the `java.library.path` system property of the server process definition.

In general, the Java virtual machine invokes findLibrary() on the class loader *xxx* that loads the class that calls System.loadLibrary(). If *xxx*.findLibrary() fails, the Java virtual machine attempts to find the library using the JVM class loader, which searches the JVM library path. If the library cannot be found, the Java virtual machine creates an UnsatisfiedLinkError exception.

Thus, if a WebSphere class loader is intended to find a native library `myNativeLib`, the library must be visible on the `nativelibpath` of the class loader that loads the class that calls System.loadLibrary(`myNativeLib`). This practice is necessary or desirable in the following situation:
- Shared libraries have a **Native Library Path** in their configuration. Because shared libraries enable the versioning of application-specific libraries, consider specifying the paths to any native libraries used by the shared library code in the shared library configuration.

Ensure that the correct WebSphere class loader loads the class that calls System.loadLibrary() and that the native library is visible on the **Native Library Path** setting.

The native library is already loaded.

This condition can result from either of the following errors:

User error

Check for multiple calls to System.loadLibrary and remove any extraneous calls.

Error when an application restarts

Invoke System.loadLibrary() within the static initialization of a class within a server-scoped shared library:

1. Remove all instances of the System.loadLibrary(*MyNativeLib*) call from the application source code.
2. Create a new class named `LibLoader` that calls System.loadLibrary(*MyNativeLib*) within a static block. For example:

```
public class LibLoader {
   static {System.loadLibrary(MyNativeLib);}
   public LibLoader();
}
```

3. Create a server-scoped shared library.
   a. Create a shared library, set its scope to **server**, and name it `MySharedLib`.
   b. Go to the settings page for the shared library.
   c. Set **Classpath** to the path containing the class LibLoader.
   d. Set **Native Library Path** to the path containing the native library *MyNativeLib*.
   e. Create a class loader on the server hosting the application.
   f. Associate MySharedLib to the new server class loader. Refer to step 2 in the Associating shared libraries with servers section of the *Administering applications and their environment* PDF book.
4. Save your changes.
5. Redeploy the application and rerun the scenario.

For more information about invoking, creating, and managing shared libraries, read Managing shared libraries in the *Administering applications and their environment* PDF book.

Classes within server-scoped libraries are loaded once for each server lifecycle, ensuring that the native library required by the application is loaded once for each Java virtual machine, regardless of the application's life cycle.

A dependent native library was used.

Dependent native libraries must be found or loaded by the JVM class loader. That is, if a native library *NL* is dependent on another native library, *DNL*, the JVM class loader must find DNL on the Java library path. This is because the JVM runs native code when loading NL; when it encounters the dependency on DNL, the JVM native code can call only to the JVM class loader to resolve the dependency. A WebSphere class loader cannot load a dependent native library.

Modify the platform-specific environment variable defining the Java library path (LIBPATH) to include the path containing the unresolved native library.

Class loader viewer service settings

Use this page to configure the server to start the class loader viewer service when the server starts. The Class Loader Viewer helps you diagnose problems with class loaders.

To view this administrative console page, click **Servers > Application servers > Class Loader Viewer Service**.

Class loaders find and load class files. For a deployed application to run properly, the class loaders that affect the application and its modules must be configured so that the application can find the files and resources that it needs. Diagnosing problems with class loaders can be complicated and time-consuming. To diagnose and fix the problems more quickly, enable the class loader viewer service on this page and then use the console Class Loader Viewer to examine class loaders and the classes loaded by each class loader. Click **Troubleshooting > Class Loader Viewer** to access the Class Loader Viewer in the console.

Enable service at server startup

Specifies whether or not the server attempts to start the class loader viewer service when the server starts.

The default is not to start the class loader viewer service.

Enterprise application topology

Use this page to see where modules reside in a topology of enterprise applications. Knowing where a module resides helps you to determine which class loader loaded a module and to diagnose problems with class loaders.

To view this administrative console page, click **Troubleshooting > Class Loader Viewer**. This page lists all installed applications and their modules in a tree view. The modules can be Web modules (`.war` files) or enterprise bean (EJB) modules (`.jar` files).

When deploying an application to a server or starting an application, you might encounter problems related to class loaders. Use the console pages accessed from this page to troubleshoot errors such as the following:

- ClassCastException
- ClassNotFoundException
- NoClassDefFoundException
- UnsatisfiedLinkError

You can use the Class Loader Viewer console pages without having to restart or manipulate the application.

Enterprise Applications Topology

Displays a tree hierarchy of applications installed on a server and lists the module files in the class paths of the applications.

Expand the hierarchy for an application to see what Web modules (`.war` files) and EJB modules (`.jar` files) are in the application class path.

Click on a module name to examine the class loaders of the module.

Class loader viewer settings

Use this page to examine the class loaders visible to a Web module (`.war` file) or enterprise bean (`.ejb` file) in an installed enterprise application. This page helps you to determine which class loaders loaded files of a module and to diagnose problems with class loaders.

To view this administrative console page, click **Troubleshooting > Class Loader Viewer >** *module_name*.

To learn more about classes used by the module and their class loaders, click a button:

Button	Resulting action
Export	Opens a dialog that enables you to view or save the Class Loader information on this page in an XML file.

Button	Resulting action
Table View	Displays the View page, which provides information about class loaders visible to the module in an HTML table format for each class loader. Such information includes: **Delegation** Whether the class loader delegates a load operation to its immediate parent before searching its local classpath for a class or resource **Classpath** The local classpath, which includes the paths over which the class loader searches for classes and resources, excluding the classpaths of any parent class loaders. **Classes** The names of classes loaded by the class loader
Search	Displays the Search page, on which you can search class loaders for the following: • Specific strings • Specific `.jar` files • The names of files in a specific directory • The names of files loaded by a specific class loader

Class Loader

Displays a tree hierarchy of class loaders that affect the loading of classes used by the Web or EJB module.

Expand the hierarchy of class loaders to view the following:

- Class loader names
- Arrows that point upwards beside class loader names, indicating that requests can go to a parent class loader only and not go to a child class loader
- The names of classes that are loaded by a class loader
- The paths of property files and `.jar` files used by the classes

The following class loaders might be in the hierarchy:

Class loader name	Description
JDK Extension Loader	The JDK extensions class loader is a composite class loader that is comprised of the Java virtual machine (JVM) bootstrap class loader, the JVM extensions class loader and the JVM system classloader, which load the core SDK classes and resources as well as classes and resources visible on the JVM classpath.
WAS Extension Class Loader	The WAS Extension Class Loader loads the WebSphere Application Server classes, standalone resource classes, custom service classes, and custom registry classes. At bootstrap, this class loader uses the ws.ext.dirs system property to determine the path that is used to load classes. Each directory in the `ws.ext.dirs` class path and every `.jar` file or `.zip` file in these directories is added to the class path used by this class loader.
WAS Compound Class Loader	The WAS Compound Class Loaders load classes and resources of application (EAR) modules, Web (WAR) modules, and server-associated shared libraries. Under default class loader policies, an instance of a WAS Compound Class Loader exists for each running EAR and WAR module and for each class loader defined in the server configuration.

Click on **Classes** to view a list of classes loaded by a class loader.

The Class Loader Viewer service must be enabled to view the list of classes.

Search settings

Use this page to search for information about class loaders visible to a Web module (.war file) or enterprise bean (.ejb file) in an installed enterprise application. This page helps you diagnose problems with class loaders.

To view this administrative console page, click **Troubleshooting** > **Class Loader Viewer** > *module_name* > **Search**.

On the Search page, you can search class loaders for the following:

- Specific strings
- Specific .jar files
- The names of files in a specific directory
- The names of files loaded by a specific class loader

Search type

Specifies the type of items in which to search for the string.

Search type	Instructions and resulting action
Class/Package	In the **Search term(s)** field, type a class name or package name. After you select this search type and click **Go**, the program searches class loaders for a class or package name. The program displays a list of classes and packages that have the string in their name.
JAR/Directory	In the **Search term(s)** field, type a .jar file name or directory name. After you select this search type and click **Go**, the program searches class loaders for a .jar file or directory name. The program displays a list of .jar files that have the string in their name and of all files in directories that have the string in their name.

Search term(s)

Specifies the string to be found in the items searched.

The search is case-sensitive. If the search string is classname, the string *ClassName* is not found.

The search matches the entire string. If the search type is **Jar/Directory** and the search string is C:/WebSphere/AppServerd0603.185/java/jre/lib/ext/CmpCrmf.jar, the entire path of the JAR file is matched. If the search type is **Jar/Directory** and the search string is Cmp, the string Cmp is not found.

The search supports limited regular expressions. It supports the wildcard characters asterisk (*), question mark (?), and percent sign (%). The wildcard characters * and % match zero or more characters; ? matches exactly one character.

Search string	Resulting matches
Cmp	Items that have Cmp in their name
Cmp.jar	Items that have Cmp in their name and that end in .jar
%Cmp%	Items that have Cmp in their name
%Cmp%.jar	Items that have Cmp in their name and that end in .jar
*Cmp?rmf.jar	Items that have a name with any characters before Cmp, then any one character, and then rmf.jar

The search supports full regular expressions if the value for the search string starts and ends with a forward slash (/).

Search string	Resulting matches
/.*Cmp.*/	Items that contain any character before and after Cmp in their name

Search string	Resulting matches
/.*Cmp.*\.jar/	Items that have Cmp in their name and that end in .jar
/.*Cmp?rmf\.jar/	Items that have a name with any characters before Cmp, then any one character, and then rmf.jar
/.*\d\.jar/	Items with a name that ends in a number followed by .jar

Diagnosing problems with message logs

WebSphere Application Server can write system messages to several general purpose logs, including JVM, process, and IBM service logs, which can be examined for problem determination.

The JVM logs are created by redirecting the System.out and System.err streams of the JVM to independent log files. WebSphere Application Server writes formatted messages to the System.out stream. In addition, applications and other code can write to these streams using the print() and println() methods defined by the streams. Some Developer Kit built-ins such as the printStackTrace() method on the Throwable class can also write to these streams. Typically, the System.out log is used to monitor the health of the running application server. The System.out log can be used for problem determination, but it is recommended to use the IBM Service log and the advanced capabilities of the Log and Trace Analyzer instead. The System.err log contains exception stack trace information that is useful when performing problem analysis.

Important: This Log and Trace Analyzer tool is different from the Log and Trace Analyzer tool available on the WebSphere® Application Server Toolkit CD-ROM. For information on the WebSphere® Application Server Toolkit Log and Trace Analyzer, refer to the *Detecting and analyzing runtime problems* documentation provided with the WebSphere® Application Server Toolkit.

Because each application server represents a JVM, there is one set of JVM logs for each application server and all of its applications located by default in the following directory:

- *profile_root*/logs/*server_name*

In the case of a WebSphere Application Server Network Deployment configuration, JVM logs are also created for the deployment manager and each node agent because they also represent JVMs.

The process logs are created by redirecting the STDOUT and STDERR streams of the process to independent log files. Native code, including the Java virtual machine (JVM) itself, writes to these files. As a general rule, WebSphere Application Server does not write to these files. However, these logs can contain information relating to problems in native code or diagnostic information written by the JVM.

As with JVM logs, there is a set of process logs for each application server, since each JVM is an operating system process, and in the case of a WebSphere Application Server Network Deployment configuration, a set of process logs for the deployment manager and each node agent.

The IBM service log contains both the WebSphere Application Server messages that are written to the System.out stream and some special messages that contain extended service information that is normally not of interest, but can be important when analyzing problems. There is one service log for all WebSphere Application Server JVMs on a node, including all application servers. The IBM Service log is maintained in a binary format and requires a special tool to view. This viewer, the Log and Trace Analyzer, provides additional diagnostic capabilities. In addition, the binary format provides capabilities that are utilized by IBM support organizations.

In addition to these general purpose logs, WebSphere Application Server contains other specialized logs that are specific to a particular component or activity. For example, the HTTP server plug-in maintains a special log. Normally, these logs are not of interest, but you might be instructed to examine one or more of these logs while performing specific problem determination procedures. For details on how and when to

view the plug-in log, see the Accessing a Web resource through the application server and bypassing the HTTP server subsection of the A Web resource does not display topic.

Sometimes server and application problems can be diagnosed by examining log output from the WebSphere Application Server.

Determine which type of logs you would like to implement:

- JVM logs
- Process logs
- IBM service logs

Viewing JVM logs

The Java virtual machine (JVM) logs are written as plain text files. Therefore there are no special requirements to view these logs.

Use either of two techniques to view the JVM logs for an application server:

- Use the administrative console, which also supports viewing the JVM logs from a remote machine.
- Use a text editor on the machine where the logs are stored.

1. View the JVM logs from the administrative console.
   a. Start the administrative console.
   b. Click **Troubleshooting > Logs and Trace** in the console navigation tree. To view the logs for a particular server, click on the server name to select it, then click **JVM Logs**.
   c. Select the runtime tab.
   d. Click **View** corresponding to the log you want to view.
2. View the JVM logs from the machine where they are stored.
   a. Go to the machine where the logs are stored.
   b. Navigate to the *profile_root*/logs/*server_name* directory and select SystemOut.log or SystemErr.log.
   c. Open the file in a text editor or drag and drop the file into an editing and viewing program.

JVM log interpretation

The JVM logs contain print data written by applications. The application can write this data directly in the form of `System.out.print()`, `System.err.print()`, or other method calls. The application can also write data indirectly by calling a JVM function, such as an `Exception.printStackTrace()`. In addition, the `System.out` JVM log contains system messages written by the WebSphere Application Server.

You can format application data to look like WebSphere Application Server system messages by using the Installed Application Output field of the JVM Logs properties panel, or as plain text with no additional formatting. WebSphere Application Server system messages are always formatted. Depending on how the JVM log is configured, formatted messages can be written to the JVM logs in either basic or advanced format.

Message formats

Formatted messages are written to the JVM logs in one of two formats:

Basic Format
The format used in earlier versions of WebSphere Application Server.

Advanced Format
Extends the basic format by adding information about an event, when possible.

Basic and advanced format fields

Basic and Advanced Formats use many of the same fields and formatting techniques. The various fields that may be found in these formats follow:

TimeStamp
The timestamp is formatted using the locale of the process where it is formatted. It includes a fully qualified date (for example YYMMDD), 24 hour time with millisecond precision and a time zone.

ThreadId
An 8 character hexadecimal value generated from the hash code of the thread that issued the message.

ThreadName
The name of the Java thread that issued the message or trace event.

ShortName
The abbreviated name of the logging component that issued the message or trace event. This is typically the class name for WebSphere Application Server internal components, but can be some other identifier for user applications.

LongName
The full name of the logging component that issued the message or trace event. This is typically the fully qualified class name for WebSphere Application Server internal components, but can be some other identifier for user applications.

EventType
A one character field that indicates the type of the message or trace event. Message types are in upper case. Possible values include:

- **F** A Fatal message.
- **E** An Error message.
- **W** A Warning message.
- **A** An Audit message.
- **I** An Informational message.
- **C** An Configuration message.
- **D** A Detail message.
- **O** A message that was written directly to `System.out` by the user application or internal components.
- **R** A message that was written directly to `System.err` by the user application or internal components.
- **Z** A placeholder to indicate the type was not recognized.

ClassName
The class that issued the message or trace event.

MethodName
The method that issued the message or trace event.

Organization
The organization that owns the application that issued the message or trace event.

Product
The product that issued the message or trace event.

Component
The component within the product that issued the message or trace event.

Basic format

Message events displayed in basic format use the following format. The notation <name> indicates mandatory fields that will always appear in the basic format message. The notation [name] indicates optional or conditional fields that will be included if they can be determined.

```
<timestamp><threadId><shortName><eventType>[className][methodName]<message>
```

Advanced format

Message events displayed in advanced format use the following format. The notation <name> is used to indicate mandatory fields that will always appear in the advanced format for message entries. The notation [name] is used to indicate optional or conditional fields that will be included if they can be determined.

```
<timestamp><threadId><eventType><UOW><source=longName>[className]
[methodName]<Organization><Product><Component>
[thread=threadName]<message>
```

Configuring the JVM logs

Use the administrative console to configure the JVM logs for an application server. Configuration changes for the JVM logs that are made to a running application server are not applied until the next restart of the application server.

1. Start the administrative console
2. Click **Troubleshooting > Logging and Tracing**, then click ***server* > JVM Logs**.
3. Select the Configuration tab.
4. Scroll through the panel to display the attributes for the stream to configure.
5. Change the appropriate configuration attributes and click **Apply**.
6. Save your configuration changes.

Java virtual machine (JVM) log settings

Use this page to view and modify the settings for the Java virtual machine (JVM) `System.out` and `System.err` logs.

To view this administrative console page, click **Troubleshooting > Logs and Trace >***server name* **> JVM Logs**.

View and modify the settings for the Java Virtual Machine (JVM) System.out and System.err logs for this managed process. The JVM logs are created by redirecting the System.out and System.err streams of the JVM to independent log files. The System.out log is used to monitor the health of the running application server. The System.err log contains exception stack trace information that is useful when performing problem analysis. There is one set of JVM logs for each application server and all of its applications. JVM logs are also created for the deployment manager and each node manager. Changes on the Configuration panel will apply when the server is restarted. Changes on the Runtime panel will apply immediately.

File Name:

Specifies the name of one of the log file described on this page.

The first file name field specifies the name of the `System.out` log. The second file name field specifies the name of the `System.err` file.

Press the **View** button on the Runtime tab to view the contents of a selected log file.

The file name specified for the `System.out` log or the `System.err` log must have one of the following values:

filename

The name of a file in the file system. It is recommended that you use a fully qualified file name. If the file name is not fully qualified, it is considered to be relative to the current working directory for the server. Each stream must be configured with a dedicated file. For example, you cannot redirect both `System.out` and `System.err` to the same physical file.

If the directory containing the file already exists, the user ID under which the server is running requires read/write access to the directory. If the directory does not exist, it will be created with the proper permissions. The user id under which the server is running must have authority to create the directory.

console
This is a special file name used to redirect the stream to the corresponding process stream. If this value is specified for `System.out`, the file is redirected to `stdout`. If this value is specified for `System.err`, the file is redirected to `stderr`.

none Discards all data written to the stream. Specifying **none** is equivalent to redirecting the stream to dev/null on an operating system such as AIX of Linux.

The default path for *filename* is the value of the variable SERVER_LOG_ROOT. To see the value of the SERVER_LOG_ROOT variable:

1. On the administrative console, select **Environment > WebSphere Variables**
2. Click on the **Server** radio button, and then click **Apply**. The value of the *SERVER_LOG_ROOT* variable appears in the resulting list.

To change the value of *SERVER_LOG_ROOT*:

1. Select SERVER_LOG_ROOT
2. Enter a new path in the Value field
3. Click Apply
4. Save the configuration. You will have to restart the server for the change to take effect.

You can also change the location and name of the ${SERVER_LOG_ROOT}/SystemOut.log and ${SERVER_LOG_ROOT}/SystemErr.log files to any other absolute path and filename (for example, /tmp/myLogfile.log).

File formatting:

Specifies the format to use in saving the `System.out` file.

Log file rotation: Use this set of configuration attributes to configure the `System.out` or `System.err` log file to be self-managing.

A self-managing log file writes messages to a file until reaching either the time or size criterion. At the specified time or when the file reaches the specified size, logging temporarily suspends while the log file rolls over, which involves closing and renaming the saved file. The new saved file name is based on the original name of the file plus a timestamp qualifier that indicates when the renaming occurs. Once the renaming completes, a new, empty log file with the original name reopens and logging resumes. All messages remain after the log file rollover, although a single message can split across the saved and the current file.

You can only configure a log to be self-managing if the corresponding stream is redirected to a file.

File Size
Click this attribute for the log file to manage itself based on its file size. Automatic roll over occurs when the file reaches the specified size you specify in the maximum size field.

Maximum Size
Specify the maximum size of the file in megabytes. When the file reaches this size, it rolls over.

This attribute is only valid if you click File size.

Time Click this attribute for the log file to manage itself based on the time of day. At the time specified in the start time field, the file rolls over.

Start Time
Specify the hour of the day, from 1 to 24, when the periodic rollover algorithm starts for the first time after an Application Server restart. The algorithm loads at Application Server startup. Once

started at the (start time field) hour, the rollover algorithm rolls the file every (repeat time field) hours. This rollover pattern continues without adjustment until the Application Server stops.

Note: The rollover always occurs at the beginning of the specified hour of the day. The first hour of the day, which starts at 00:00:00 (midnight), is hour 1 and the last hour of the day, which starts at 23:00:00, is hour 24. Therefore, if you want log files to roll over at midnight, set the start time to 1.

Repeat time
Specifies the number of hours after which the log file rolls over. Valid values range from 1 to 24.

Configure a log file to roll over by time, by size, or by time and size. Click **File Size** and **Time** to roll the file at the first matching criterion. For example, if the repeat time field is 5 hours and the maximum file size is 2 MB, the file rolls every 5 hours, unless it reaches 2 MB before the interval elapses. After the size rollover, the file continues to roll at each interval.

Maximum Number of Historical Log Files: Specifies the number of historical (rolled) files to keep. The stream writes to the current file until it rolls. At rollover, the current file closes and is saved as a new name consisting of the current name plus the rollover timestamp. The stream then reopens a new file with the original name to continue writing. The number of historical files grows from zero to the value of the maximum number of historical files field. The next rollover deletes the oldest historical file.

Installed Application Output: Specifies whether `System.out` or `System.err` print statements issued from application code are logged and formatted.

Show application print statements
Click this field to show messages that applications write to the stream using **print** and **println** stream methods. WebSphere Application Server system messages always appear.

Format print statements
Click this field to format application print statement like WebSphere Application Server system messages.

Process logs

WebSphere Application Server processes contain two output streams that are accessible to native code running in the process. These streams are the stdout and stderr streams. Native code, including Java virtual machines (JVM), might write data to these process streams. In addition, JVM provided System.out and System.err streams can be configured to write their data to these streams also.

By default, the stdout and stderr streams are redirected to log files at application server startup, which contain text written to the stdout and stderr streams by native modules (*SRVPGMs, .dlls, .exes, UNIX libraries, and other modules). By default, these files are stored as *profile_root*/logs/*server_name*/native_stderr.log and *profile_root*/logs/native_stdout.log.

Configuring the service log

By default, the service log is shared among all server processes for a node. The configuration values for the service log are inherited by each server process from the node configuration. You can configure a separate service log for each server process by overriding the configuration values at the server level.

1. Start the administrative console.
2. Click **Troubleshooting > Logs and Trace > *server_name* > IBM Service Logs**.
3. Select the **Enable** box to enable the service log, clear the check box to disable the log.
4. Set the name for the service log.

   The default name is *profile_root*/logs/activity.log. If the name is changed, the run time requires write access to the new file, and the file must use the .log extension.
5. Set the maximum file size. Specifies the number of megabytes to which the file can grow. When the file reaches this size, it wraps, replacing the oldest data with the newest data.

6. Set the message filter level to the desired state.
7. Save the configuration.
8. Restart the server to apply the configuration changes.

IBM service log settings

To view this administrative console page, click **Troubleshooting > Logs and Trace** > *server name* > **IBM Service Logs**.

Use this panel to configure the IBM service log, also known as the activity log. The IBM service log contains both the WebSphere Application Server messages that are written to the System.out stream and some special messages that contain extended service information that can be important when analyzing problems. There is one service log for all WebSphere Application Server Java virtual machines (JVMs) on a node, including all application servers and their node agent (if present). A separate activity log is created for a deployment manager in its own logs directory. The IBM Service log is maintained in a binary format. Use the Log and Trace Analyzer or Showlog tool to view the IBM service log.

Enable service log:

Specifies creation of a log file by the IBM Service log.

File Name:

Specifies the name of the file used by the IBM Service log.

Maximum File Size:

Specifies the maximum size in megabytes of the service log file. The default value is 2 megabytes.

When this size is reached, the service log wraps in place. Note that the service log does not roll over to a new log file like the JVM logs.

Enable Correlation ID:

Specifies the generation of a correlation ID that is logged with each message.

You can use the correlation ID to correlate activity to a particular client request, or correlate activities on multiple application servers.

Viewing the service log

Service logs are logs written in a binary format. You cannot view a service log directly using a text editor. You should never directly edit the service log, as doing so will corrupt the log.

To move a service log from one machine to another, you must use a mechanism like FTP, which supports binary file transfer. You can view a service log in two ways:

- It is recommended that you use the Log and Trace Analyzer tool to view the service log. This tool provides interactive viewing and analysis capability that is helpful in identifying problems.
- If you are unable to use the Log and Trace Analyzer tool, use the Showlog tool to convert the contents of the service log to a text format that you can then write to a file or dump to the command shell window.

Run the showlog script to view the contents of the service log as described in the following procedure.

1. Open a shell window on the machine where the service log resides.
2. Change the directory to *app_server_root*/bin where *app_server_root* is the fully qualified path where the WebSphere Application Server product is installed.
3. Run the showlog script.

```
showlog
```

4. Run the following showlog script with no parameters to display usage instructions.

   ```
   showlog
   ```

5. Format and write the service log contents to a file.

   ```
   showlog service_log_filename output_filename
   ```

   If the service log is not in the default location, you must fully qualify the *service_log_filename*

Message reference

Messages logged by application server components and associated IBM products start with a unique message identifier that indicates the component or application that issued the message.

You can log WebSphere Application Server system messages from a variety of sources, including application server components and applications. The message identifier can be either 8 or 9 characters in length and has the form:

```
CCCC1234X
```

where:

CCCC is a four character alphabetic component or application identifier.
1234 is a four character numeric identifier used to identify the specific message for that component.
X is an optional alphabetic severity indicator. (I=Informational, W=Warning, E=Error)

To view the messages generated by WebSphere Application Server components, select the **Reference** view of the information center navigation and expand **Troubleshooter** > **Messages**.

CORBA minor codes

Applications that use CORBA services generate minor codes to indicate the underlying cause of a failure. These codes are written to the exception stack. Look for ″minor code″ in the exception stack to locate these exceptions.

Overview

Common Object Request Broker Architecture (CORBA) is an industry-wide standard for object-oriented communication between processes, which is supported in several programming languages. Several subcomponents of WebSphere Application Server use CORBA to communicate across processes.

When a CORBA process fails, that is a request from one process to another cannot be sent, completed, or returned, a high-level exception is created, such as TransactionRolledBackException: CORBA TRANSACTION_ROLLEDBACK.

Minor codes that are used by WebSphere Application Server components

Range	Related subcomponent	Where to find details
0x49424300-0x494243FF	Security	“Security components troubleshooting tips” on page 322
0x49421050-0x4942105F, 0x49421070-0x4942107F	ORB services	“Object request broker troubleshooting tips” on page 378
0x4f4d and above	Standard CORBA exceptions	http://www.omg.org

Range	Related subcomponent	Where to find details
0x49421080-0x4942108F	Naming services	See the *Reference: Generated API documentation* reference section for information about the ws.code.naming.src.com.ibm.websphere.naming.WsnCorbaMinorCodes class.
0x49421080-0x4942108F	Workload Management	See the *Reference: Generated API documentation* reference section for information about the com.ibm.websphere.wlm.WsCorbaMinorCodes class.

Configuring the hang detection policy

A common error in J2EE applications is a hung thread. A hung thread can result from a simple software defect (such as an infinite loop) or a more complex cause (for example, a resource deadlock). System resources, such as CPU time, might be consumed by this hung transaction when threads run unbounded code paths, such as when the code is running in an infinite loop. Alternately, a system can become unresponsive even though all resources are idle, as in a deadlock scenario. Unless an end user or a monitoring tool reports the problem, the system may remain in this degraded state indefinitely.

The hang detection option for WebSphere Application Server is turned on by default. You can configure a hang detection policy to accommodate your applications and environment so that potential hangs can be reported, providing earlier detection of failing servers. When a hung thread is detected, WebSphere Application Server notifies you so that you can troubleshoot the problem.

Using the hang detection policy, you can specify a time that is too long for a unit of work to complete. The thread monitor checks all managed threads in the system (for example, Web container threads and object request broker (ORB) threads) . Unmanaged threads, which are threads created by applications, are not monitored. For more information read “Hung threads in J2EE applications” on page 109.

The thread hang detection option is enabled by default. To adjust the hang detection policy values, or to disable hang detection completely:

1. From the administrative console, click **Servers** > **Application Servers** > *server_name*
2. Under Server Infrastructure, click **Administration** > **Custom Properties**
3. Click **New**.
4. Add the following properties:

```
Name: com.ibm.websphere.threadmonitor.interval
Value:  The frequency (in seconds) at which managed threads in the selected
        application server will be interrogated.
Default: 180 seconds (three minutes).

Name: com.ibm.websphere.threadmonitor.threshold
Value:   The length of time (in seconds) in which a thread can be active
         before it is considered hung.  Any thread that is detected as
         active for longer than this length of time is reported as hung.
Default: The default value is 600 seconds (ten minutes).

Name:com.ibm.websphere.threadmonitor.false.alarm.threshold
Value: The number of times (T) that false alarms can occur
       before automatically increasing the threshold. It is possible that a
       thread that is reported as hung eventually completes its work,
       resulting in a false alarm.  A large number of these events indicates
       that the threshhold value is too small. The hang detection facility can
       automatically respond to this situation: For every T false alarms, the
       threshold T is increased by a factor of 1.5. Set the value to
       zero (or less) to disable the automatic adjustment.
Default: 100
```

To disable the hang detection option, set the **com.ibm.websphere.threadmonitor.interval** property to less than or equal to zero.

5. Click **Apply**.
6. Click **OK**.
7. Save the changes and make sure a file synchronization is performed before restarting the servers.
8. Restart the Application Server for the changes to take effect.

Hung threads in J2EE applications

WebSphere Application Server monitors thread activity and performs diagnostic actions if one has become inactive.

When WebSphere detects that a thread has been active longer than the time defined by the thread monitor threshold, the application server takes the following actions:

- Logs a warning in the WebSphere Application Server log that indicates the name of the thread that is hung and how long it has already been active. The following message is written to the log:

  ```
  WSVR0605W: Thread threadname has been active for
  hangtime and may be hung.  There are totalthreads
  threads in total in the server that may be hung.
  ```

  where: *threadname* is the name that appears in a JVM thread dump, *hangtime* gives an approximation of how long the thread has been active and *totalthreads* gives an overall assessment of the system threads.
- Issues a Java Management Extensions (JMX) notification. This notification enables third-party tools to catch the event and take appropriate action, such as triggering a JVM thread dump of the server, or issuing an electronic page or e-mail. The following JMX notification events are defined in the com.ibm.websphere.management.NotificationConstants class:
  - TYPE_THREAD_MONITOR_THREAD_HUNG This event is triggered by the detection of a (potentially) hung thread.
  - TYPE_THREAD_MONITOR_THREAD_CLEAR This event is triggered if a thread that was previously reported as hung completes its work. Consult the section on false alarms for more information.
- Triggers changes in the performance monitoring infrastructure (PMI) data counters. These PMI data counters are used by various tools, such as the Tivoli Performance Viewer, to provide a performance analysis.
- Triggers changes in the performance monitoring infrastructure (PMI) data counters. These PMI data counters are used by various tools, such as the Tivoli Performance Viewer (TPV), to provide a performance analysis.

For additional information about performance monitoring and Tivoli Performance Viewer, see the chapter Monitoring performance with Tivoli Performance Viewer (TPV) in the *Tuning guide* PDF book

False Alarms

If the work actually completes, a second set of messages, notifications and PMI events is produced to identify the false alarm. The following message is written to the log:

```
WSVR0606W: Thread threadname was previously reported to be
hung but has completed. It was active for approximately hangtime.
There are totalthreads threads in total in the server that still
may be hung.
```

where *threadname* is the name that appears in a JVM thread dump, *hangtime* gives an approximation of how long the thread has been active and *totalthreads* gives an overall assessment of the system threads.

Automatic adjustment of the hang time threshold

If the thread monitor determines that too many false alarms are issued (determined by the number of pairs of hang and clear messages), it can automatically adjust the threshold. When this adjustment occurs, the following message is written to the log:

```
WSVR0607W: Too many thread hangs have been falsely reported.  The hang
threshold is now being set to thresholdtime.
```

where: *thresholdtime* is the time (in seconds) in which a thread can be active before it is considered hung.

You can prevent WebSphere Application Server from automatically adjusting the hang time threshold. See "Configuring the hang detection policy" on page 108

Hang detection policy of a running server

The hang detection policy affects how the application server responds to a thread that is not being processed correctly.

You can adjust the thread monitor settings by using the wsadmin scripting interface. These changes take effect immediately, but do not persist to the server configuration, and are lost when the server is restarted. The following script provides an example of how to adjust the properties for the thread monitor using the wsadmin tool:

```
# Read in the interval, threshold, false alarm from the command line
set interval [lindex $argv 0]
set threshold [lindex $argv 1]
set adjustment [lindex $argv 2]

# Get the object name of the server you want to change the values on
set server [$AdminControl completeObjectName "type=Server,*"]

# Read in the interval and print to the console
set i [$AdminControl getAttribute $server threadMonitorInterval]

# Read in the threshold and print to the console
set t [$AdminControl getAttribute $server threadMonitorThreshold]

# Read in the false alarm adjustment threshold and print to the console
set a [$AdminControl getAttribute $server threadMonitorAdjustmentThreshold]

# Set the new values using the command line parameters
$AdminControl setAttribute $server threadMonitorInterval  ${interval}

$AdminControl setAttribute $server threadMonitorThreshold ${threshold}

$AdminControl setAttribute $server threadMonitorAdjustmentThreshold ${adjustment}
```

Working with trace

Use trace to obtain detailed information about running the WebSphere Application Server components, including application servers, clients, and other processes in the environment.

Trace files show the time and sequence of methods called by WebSphere Application Server base classes, and you can use these files to pinpoint the failure.

Collecting a trace is often requested by IBM technical support personnel. If you are not familiar with the internal structure of WebSphere Application Server, the trace output might not be meaningful to you.

1. Configure an output destination to which trace data is sent.
2. Enable trace for the appropriate WebSphere Application Server or application components.
3. Run the application or operation to generate the trace data.

4. Analyze the trace data or forward it to the appropriate organization for analysis.

For current information available from IBM Support on known problems and their resolution, see the IBM Support page.

IBM Support has documents that can save you time gathering information needed to resolve this problem. Before opening a PMR, see the IBM Support page.

Enabling trace on client and standalone applications

When standalone client applications (such as Java applications which access enterprise beans hosted in WebSphere Application Server) have problems interacting with WebSphere Application Server, it may be useful to enable tracing for the application. Enabling trace for client programs will cause the WebSphere Application Server classes used by those applications, such as naming-service client classes, to generate trace information. A common troubleshooting technique is to enable tracing on both the application server and client applications, and match records according to timestamp to try to understand where a problem is occurring.

1. To enable trace for the WebSphere Application Server classes in a client application, add the system properties shown in the following example to the startup script or command of the client application. The location of the output and the classes and detail included in the trace follow the same rules as for adding trace to WebSphere Application Servers. For example, trace the standalone client application program named com.ibm.sample.MyClientProgram, you would enter the following command:

   ```
   java -DtraceSettingsFile=MyTraceSettings.properties
   -Djava.util.logging.manager=com.ibm.ws.bootstrap.WsLogManager
   -Djava.util.logging.configureByServer=true com.ibm.samples.MyClientProgram
   ```

   The file identified by *filename* must be a properties file placed in the classpath of the application client or stand-alone process. An example file is provided in

   - *profile_root*/properties/TraceSettings.properties

   You cannot use the **-DtraceSettingsFile=TraceSettings.properties** property to enable tracing of the ORB component for thin clients. ORB tracing output for thin clients can be directed by setting **com.ibm.CORBA.Debug.Output = debugOutputFilename** parameter in the command line.

   The java.util.logging.manager and java.util.logging.configureByServer system properties configure Java logging to use a WebSphere Application Server-specific LogManager class and to use the configuration from the file specified by the traceSettingsFile property. The default Java Logging properties file, located in the Java Runtime Environment (JRE), will not be applied.
2. You can configure the MyTraceSettings.properties file to send trace output to a file using the traceFileName property. Specify one of two options:
   - The fully qualified name of an output file. For example, traceFileName=c:\\MyTraceFile.log. You must specify this property to generate visible output.
   - stdout. When specified, output is written to System.out.
3. You can also specify a trace string for writing messages with the Trace String property, Specify a startup trace specification similar to that available on the server. For your convenience, you can enter multiple individual trace strings into the trace settings file, one trace string per line.

Here are the results of using each optional property setting:
- Specify a valid setting for the traceFileName property without a trace string to write messages to the specified file or System.out only.
- Specify a trace string without a traceFileName property value to generate no output.
- Specify both a valid traceFileName property and a trace string to write both message and trace entries to the location specified in the traceFileName property.

Tracing and logging configuration

You can configure tracing and logging settings to help diagnose problems or evaluate system performance.

You can configure the application server to start in a trace-enabled state by setting the appropriate configuration properties. You can only enable trace for an application client or standalone process at process startup.

In WebSphere Application Server, V6 and later, a logging infrastructure, extending Java Logging, is used. This results in the following changes to the configuration of the logging infrastructure in WebSphere Application Server:

- Loggers defined in Java logging are equivalent to, and configured in the same way as, trace components introduced in previous versions of WebSphere Application Server. Both are referred to as ″components.″
- Both Java logging levels and WebSphere Application Server levels can be used. The following is a complete list of valid levels, ordered in ascending order of severity:
  1. all
  2. finest or debug
  3. finer or entryExit
  4. fine or event
  5. detail
  6. config
  7. info
  8. audit
  9. warning
  10. severe or error
  11. fatal
  12. off
- Setting the logging and tracing level for a component to all will enable all the logging for that component. Setting the logging and tracing level for a component to off will disable all the logging for that component.
- You can only configure a component to one level. However, configuring a component to a certain level enables it to perform logging on the configured level and any higher severity level.
- Several levels have equivalent names: finest is equivalent to debug; finer is equivalent to entryExit; fine is equivalent to event; severe is equivalent to error.

Java Logging does not distinguish between tracing and message logging. However, previous versions of WebSphere Application Server have made a clear distinction between those kind of messages. In WebSphere Application Server, V6 and later, the differences between tracing and message logging are as follows:

- Tracing messages are messages with lower severity (for example, tracing messages are logged on levels fine, finer, finest, debug, entryExit, or event).
- Tracing messages are generally not localized.
- When tracing is enabled, a much higher volume of messages will be produced.
- Tracing messages provide information for problem determination.

Trace and logging strings

In WebSphere Application Server, V5.1.1 and earlier, trace strings were used for configuring tracing only. Starting in WebSphere Application Server, Version 6 and later, the ″trace string″ becomes a ″logging string″; it is used to configure both tracing and message logging.

In WebSphere Application Server, V5.1.1 and earlier, the trace service for all WebSphere Application Server components is disabled by default. To request a change to the current state of the trace service, a trace string is passed to the trace service. This trace string encodes the information detailing which level of trace to enable or disable and for which components.

In all versions of WebSphere Application Server, the tracing for all components is disabled by default. To change to the current state of the tracing and message logging, a logging string must be constructed and passed to the server. This logging string specifies what level of trace or logging to enable or disable for specific components.

You can type in trace strings (or logging strings), or construct them using the administrative console. Trace and logging strings must conform to a specific grammar.

For WebSphere Application Server, V5.1.1 and earlier, the specification of this grammar is as follows:

```
TRACESTRING=COMPONENT_TRACE_STRING[:COMPONENT_TRACE_STRING]*

 COMPONENT_TRACE_STRING=COMPONENT_NAME=LEVEL=STATE[,LEVEL=STATE]*

LEVEL = all | entryExit | debug | event

STATE = enabled | disabled

COMPONENT_NAME = COMPONENT | GROUP
```

For WebSphere Application Server, V6 and later, the previous grammar is supported. However a new grammar has been added to better represent the underlying infrastructure:

```
LOGGINGSTRING=COMPONENT_LOGGING_STRING[:COMPONENT_LOGGING_STRING]*

 COMPONENT_TRACE_STRING=COMPONENT_NAME=LEVEL

LEVEL = all | (finest | debug) | (finer | entryExit) | (fine | event )
| detail | config | info | audit | warning | (severe | error) | fatal | off

COMPONENT_NAME = COMPONENT | GROUP
```

The COMPONENT_NAME is the name of a component or group registered with the trace service logging infrastructure. Typically, WebSphere Application Server components register using a fully qualified Java class name, for example com.ibm.servlet.engine.ServletEngine. In addition, you can use a wildcard character of asterisk (*) to terminate a component name and indicate multiple classes or packages. For example, use a component name of com.ibm.servlet.* to specify all components whose names begin with com.ibm.servlet. You can use a wildcard character of asterisk (*) at the end of the component or group name to make the logging string applicable to all components or groups whose names start with specified string. For example, a logging string specifying ″com.ibm.servlet.*″ as a component name will be applied to all components whose names begin with com.ibm.servlet. When an asterisk (*) is used by itself in place of the component name, the level the string specifies, will be applied to all components.

The following are examples of using an asterisk (*) in logging strings. Note that the asterisk (*) in the logging string does not need to have a period (.) in front of it. The period (.) can be used anywhere in the logging string.

- `com.ibm.ejs.ras.*=all` - enables tracing for all loggers with names starting with ″com.ibm.ejs.ras.″. If there is a logger named ″com.ibm.ejs.ras″ it will not have trace enabled.
- `com.ibm.ejs.ras*=all` - enables tracing for all loggers with names starting with ″com.ibm.ejs.ras″, such as com.ibm.ejs.ras, com.ibm.ejs.raslogger, com.ibm.ejs.ras.ManagerAdmin

Note:

- In WebSphere Application Server, V5.1.1 and earlier, you could set the level to ″all=disabled″ to disable tracing. This syntax, beginning with Version 6.0, will result in LEVEL=info; tracing will be disabled, but logging will be enabled.
- The logging string is processed from left to right. During the processing, part of the logging string might be modified or removed if the levels they configure are being overridden by another part of the logging string.
- In WebSphere Application Server, V6 and later, ″info″ is the default level. If the specified component is not present (*=xxx is not found), *=info is always implied. Any component that is not matched by the trace string will have its level set to info.
- If the logging string does not start with a component logging string specifying a level for all components, using the ″*″ in place of component name, one will be added, setting the default level for all components.
- STATE = enabled | disabled is not needed in Version 6 and later. However, if used, it has the following effect:
  - ″enabled″ sets the logging for the component specified to the level specified
  - ″disabled″ sets the logging for the component specified to one level above the level specified. The following examples illustrate the effect that disabling has on the logging level:

Logging string	Resulting logging level	Notes
com.ibm.ejs.ras=debug=disabled	com.ibm.ejs.ras=finer	debug (version 5) = finest (version 6)
com.ibm.ejs.ras=all=disabled	com.ibm.ejs.ras=info	″all=disabled″ will disable tracing; logging is still enabled.
com.ibm.ejs.ras=fatal=disabled	com.ibm.ejs.ras=off	
com.ibm.ejs.ras=off=disabled	com.ibm.ejs.ras=off	off is the highest severity

Examples of legal trace strings include:

Version 5 syntax	Version 6 syntax
`com.ibm.ejs.ras.ManagerAdmin=debug=enabled`	`com.ibm.ejs.ras.ManagerAdmin=finest`
`com.ibm.ejs.ras.ManagerAdmin=all=enabled,event=disabled`	`com.ibm.ejs.ras.ManagerAdmin=detail`
`com.ibm.ejs.ras.*=all=enabled`	`com.ibm.ejs.ras.*=all`
`com.ibm.ejs.ras.*=all=enabled:com.ibm.ws.ras=debug=enabled,entryexit=enabled`	`com.ibm.ejs.ras.*=all:com.ibm.ws.ras=finer`

Enabling trace at server startup

The diagnostic trace configuration settings for a server process determines the initial trace state for a server process. The configuration settings are read at server startup and used to configure the trace service. You can also change many of the trace service properties or settings while the server process is running.

1. Start the administrative console.
2. Click **Troubleshooting > Logging and Tracing** in the console navigation tree, then click **Server > Diagnostic Trace**.
3. Click **Configuration**.
4. Select the **Enable Log** check box to enable trace, clear the check box to disable trace.
5. Select whether to direct trace output to either a file or an in-memory circular buffer.

Note: Different components can produce different amounts of trace output per entry. Naming and security tracing, for example, produces a much higher trace output than Web container tracing. Consider the type of data being collected when you configure your memory allocation and output settings.

6. If the in-memory circular buffer is selected for the trace output set the size of the buffer, specified in thousands of entries. This is the maximum number of entries that will be retained in the buffer at any given time.
7. If a file is selected for trace output, set the maximum size in megabytes to which the file should be allowed to grow. When the file reaches this size, the existing file will be closed, renamed, and a new file with the original name reopened. The new name of the file will be based upon the original name with a timestamp qualifier added to the name. In addition, specify the number of history files to keep.
8. Select the desired format for the generated trace.
9. Save the changed configuration.
10. To enter a trace string to set the trace specification to the desired state:
    a. Click **Troubleshooting > Logging and Tracing** in the console navigation tree.
    b. Select a server name.
    c. Click **Change Log Level Details**.
    d. If **All Components** has been enabled, you might want to turn it off, and then enable specific components.
    e. Click a component or group name. For more information see “Log level settings” on page 62 If the selected server is not running, you will not be able to see individual component in graphic mode.
    f. Enter a trace string in the trace string box.
    g. Select **Apply**, then **OK**.
11. Allow enough time for the nodes to synchronize, and then start the server.

Enabling trace on a running server

You can modify the trace service state that determines which components are being actively traced for a running server by using the following procedure.

1. Start the administrative console.
2. Click **Troubleshooting > Logs and Trace** in the console navigation tree, then click **server > Diagnostic Trace**.
3. Select the **Runtime** tab.
4. Select the **Save runtime changes to configuration as well** check box if you want to write your changes back to the server configuration.
5. Change the existing trace state by changing the trace specification to the desired state.
6. Configure the trace output if a change from the existing one is desired.
7. Click **Apply**.

Managing the application server trace service

You can manage the trace service for a server process while the server is stopped and while it is running. You can specify which components to trace, where to send trace output, the characteristics of the trace output device, and which format to generate trace output in.

Modify the trace settings to help with diagnosing problems or tuning performance within certain applications. To manage the trace service for a server process:

1. Start the administrative console.
2. Click **Troubleshooting > Logging and Tracing** in the console navigation tree, then click ***server* > Diagnostic Trace**
3. If the server is running, select the **Runtime** tab.

4. For a running server, check the Save trace check box to write your changes back to the server configuration. If Save trace is not selected, the changes you make will apply only for the life of the server process that is currently running.
5. Perform the desired operation:
   a. Enter the file name and click **Dump** to dump the in-memory circular buffer.
   b. To change the trace destination from a file to the in-memory circular buffer or to a different file, or to change from the in memory circular buffer to a file, select the appropriate radio buttons, then click Apply.

      Note: Different components can produce different amounts of trace output per entry. Naming and security tracing, for example, produces a much higher trace output than Web container tracing. Consider the type of data being collected when you configure your memory allocation and output settings.
   c. To change the format in which trace output is generated, select the appropriate value from the drop-down list.

Interpreting trace output

Trace output allows administrators to examine processes in the application server and diagnose various issues.

On an application server, trace output can be directed either to a file or to an in-memory circular buffer. If trace output is directed to the in-memory circular buffer, it must be dumped to a file before it can be viewed.

On an application client or stand-alone process, trace output can be directed either to a file or to the process console window.

In all cases, trace output is generated as plain text in either basic, advanced or log analyzer format as specified by the user. The basic and advanced formats for trace output are similar to the basic and advanced formats that are available for the JVM message logs.

Basic and advanced format fields

Basic and Advanced Formats use many of the same fields and formatting techniques. The fields that can be used in these formats include:

TimeStamp
The timestamp is formatted using the locale of the process where it is formatted. It includes a fully qualified date (YYMMDD), 24 hour time with millisecond precision and the time zone.

ThreadId
An 8 character hexadecimal value generated from the hash code of the thread that issued the trace event.

ThreadName
The name of the Java thread that issued the message or trace event.

ShortName
The abbreviated name of the logging component that issued the trace event. This is typically the class name for WebSphere Application Server internal components, but may be some other identifier for user applications.

LongName
The full name of the logging component that issued the trace event. This is typically the fully qualified class name for WebSphere Application Server internal components, but may be some other identifier for user applications.

EventType
A one character field that indicates the type of the trace event. Trace types are in lower case. Possible values include:

> a trace entry of type method entry.
< a trace entry of type method exit.
1 a trace entry of type fine or event.
2 a trace entry of type finer.
3 a trace entry of type finest, debug or dump.
Z a placeholder to indicate that the trace type was not recognized.

ClassName
The class that issued the message or trace event.

MethodName
The method that issued the message or trace event.

Organization
The organization that owns the application that issued the message or trace event.

Product
The product that issued the message or trace event.

Component
The component within the product that issued the message or trace event.

Basic format

Trace events displayed in basic format use the following format:

```
<timestamp><threadId><shortName><eventType>[className][methodName]<textmessage>
               [parameter 1]
               [parameter 2]
```

Advanced formats

Trace events displayed in advanced format use the following format:

```
<timestamp><threadId><eventType><UOW><source=longName>[className][methodName]
<Organization><Product><Component>[thread=threadName]
<textMessage>[parameter 1=parameterValue][parameter 2=parameterValue]
```

Log analyzer trace format

Preserves trace information in the same format as produced by Showlog tool.

Diagnostic trace service settings

To view this page, click the following paths:

- **Troubleshooting > Logs and Trace > *server* > Diagnostic trace**

Enable Log

Enables the log service.

If this option is not selected, the following configuration properties are not passed to the application server trace service at server startup.

Trace Output

Specifies where trace output should be written. The trace output can be written directly to an output file, or stored in memory and written to a file on demand using the Dump button found on the run-time page.

Different components can produce different amounts of trace output per entry. Naming and security tracing, for example, produces a much higher trace output than Web container tracing. Consider the type of data being collected when you configure your memory allocation and output settings.

Memory Buffer

Specifies that the trace output should be written to an in-memory circular buffer. If you select this option you must specify the following parameters:

- **Maximum Buffer Size**
  - Specifies the number of entries, in thousands, that can be cached in the buffer. When this number is exceeded, older entries are overwritten by new entries.
- **Dump File Name**
  - The name of the file to which the memory buffer will be written when it is dumped. This option is only available from the Runtime tab.

File

Specifies to write the trace output to a self-managing log file. The self-managing log file writes messages to the file until the specified maximum file size is reached. When the file reaches the specified size, logging is temporarily suspended and the log file is closed and renamed. The new name is based on the original name of the file, plus a timestamp qualifier that indicates when the renaming occurred. Once the renaming is complete, a new, empty log file with the original name is reopened, and logging resumes. No messages are lost as a result of the rollover, although a single message may be split across the two files. If you select this option you must specify the following parameters:

- **Maximum File Size**
  - Specifies the maximum size, in megabytes, to which the output file is allowed to grow. This attribute is only valid if the File Size attribute is selected. When the file reaches this size, it is rolled over as described above.
- **Maximum Number of Historical Files**
  - Specifies the maximum number of rolled over files to keep.
- **File Name**
  - Specifies the name of the file to which the trace output is written.

Trace Output Format

Specifies the format of the trace output.

You can specify one of three levels for trace output:

- **Basic (Compatible)**
  - Preserves only basic trace information. Select this option to minimize the amount of space taken up by the trace output.
- **Advanced**
  - Preserves more specific trace information. Select this option to see detailed trace information for use in troubleshooting and problem determination.
- **Log analyzer trace format**
  - Preserves trace information in the same format as produced by Showlog tool.

Runtime tab

Specifies the format of the trace output.

Save changes to configuration

Save changes made on the runtime tab to the trace configuration as well. Select this box to copy run-time trace changes to the trace configuration settings as well. Saving these changes to the trace configuration will cause the changes to persist even if the application is restarted.

Trace Output

Specifies where trace output should be written. The trace output can be written directly to an output file, or stored in memory and written to a file on demand using the Dump button found on the run-time page.

Memory Buffer

Specifies that the trace output should be written to an in-memory circular buffer. If you select this option you must specify the following parameters:

- **Maximum Buffer Size**
  - Specifies the number of entries, in thousands, that can be cached in the buffer. When this number is exceeded, older entries are overwritten by new entries.
- **Dump File Name**
  - The name of the file to which the memory buffer will be written when it is dumped. This option is only available from the Runtime tab.

File

Specifies to write the trace output to a self-managing log file. The self-managing log file writes messages to the file until the specified maximum file size is reached. When the file reaches the specified size, logging is temporarily suspended and the log file is closed and renamed. The new name is based on the original name of the file, plus a timestamp qualifier that indicates when the renaming occurred. Once the renaming is complete, a new, empty log file with the original name is reopened, and logging resumes. No messages are lost as a result of the rollover, although a single message may be split across the two files. If you select this option you must specify the following parameters:

- **Maximum File Size**
  - Specifies the maximum size, in megabytes, to which the output file is allowed to grow. This attribute is only valid if the File Size attribute is selected. When the file reaches this size, it is rolled over as described above.
- **Maximum Number of Historical Files**
  - Specifies the maximum number of rolled over files to keep.
- **File Name**
  - Specifies the name of the file to which the trace output is written.
- **File Name**
  - View the file that is specified by the **File Name** parameter. This does not apply your configuration.

Select a server to configure logging and tracing

Use this page to select the server for which you want to configure logging and trace settings.

Application Servers

This page lists application servers in the cell and the nodes holding the application servers. If you are using the Network Deployment product, this panel also shows the status of the application servers. The status indicates whether a server is running, stopped, or encountering problems.

When you select an application server, a panel is displayed that will allow you to choose which log or trace task to configure for that application server.

To view this administrative console page, click **Troubleshooting** > **Logs and Trace**

Name

Specifies the logical name of the server.

Node

Specifies the name of the node for the application server.

Version

Specifies the version for the application server.

Status
Indicates whether the application server is started or stopped. (Network Deployment only)

Note that if the status is *Unavailable*, the node agent is not running in that node and you must restart the node agent before you can start the server.

Log and trace settings
Use this page to view and configure logging and trace settings for the server.

To view this administrative console page, click the following paths:

- **Troubleshooting > Logs and Trace >** *server_name*

Diagnostic Trace
The diagnostic trace configuration settings for a server process determine the initial trace state for a server process. The configuration settings are read at server startup and used to configure the trace service. You can also change many of the trace service properties or settings while the server process is running.

Java virtual machine (JVM) Logs
The JVM logs are created by redirecting the System.out and System.err streams of the JVM to independent log files. WebSphere Application Server writes formatted messages to the System.out stream. In addition, applications and other code can write to these streams using the print() and println() methods defined by the streams.

Process Logs
WebSphere Application Server processes contain two output streams that are accessible to native code running in the process. These streams are the stdout and stderr streams. Native code, including Java virtual machines (JVM), might write data to these process streams. In addition, JVM provided System.out and System.err streams can be configured to write their data to these streams also.

IBM Service Logs
The IBM service log contains both the WebSphere Application Server messages that are written to the System.out stream and some special messages that contain extended service information that is normally not of interest, but can be important when analyzing problems. There is one service log for all WebSphere Application Server JVMs on a node, including all application servers. The IBM Service log is maintained in a binary format and requires a special tool to view. This viewer, the Log and Trace Analyzer, provides additional diagnostic capabilities. In addition, the binary format provides capabilities that are utilized by IBM support organizations.

Change Log Level Details
Enter a log detail level that specifies the components, packages, or groups to trace. The log detail level string must conform to the specific grammar described in this topic. You can enter the log detail level string directly, or generate it using the graphical trace interface.

Working with troubleshooting tools

WebSphere Application Server includes a number of troubleshooting tools that are designed to help you isolate the source of problems. Many of these tools are designed to generate information to be used by IBM Support, and their output might not be understandable by the customer.

This section only discusses tools that are bundled with the WebSphere Application Server product. A wide range of tools which address a variety of problems is available from the WebSphere Application Server Technical Support Web site.

1. Select the appropriate tool for the task. For more information on the capacities of the supplied troubleshooting tools, see the relevant articles in this section.
2. Run the tool as described in the relevant article.

3. Contact IBM Support for assistance in deciphering the output of the tool. For current information available from IBM Support on known problems and their resolution, see the IBM Support page. IBM Support has documents that can save you time gathering information needed to resolve this problem. For the last minute updates, limitations, and known problems, see the Release notes. Before opening a PMR, see the Must gather page.
4. Use the IBM Support Assistant to help find and use various IBM Support resources, such as updated documentation and problem determination tools.

Gathering information with the Collector tool

The collector tool gathers information about your WebSphere Application Server installation and packages it in a Java archive (JAR) file that you can send to IBM Customer Support to assist in determining and analyzing your problem. Information in the JAR file includes logs, property files, configuration files, operating system and Java data, and the presence and level of each software prerequisite.

There are two ways to run the collector tool. Run the collector tool to collect summary data or to traverse the system to gather relevant files and command results. The collector tool produces a Java archive (JAR) file of information needed to determine and solve a problem. The collector summary option produces a lightweight collection of version and other information that is useful when first reporting the problem to IBM Support.

There are two phases of using the collector tool. The first phase runs the collector tool on your WebSphere Application Server product and produces a Java archive (JAR) file . The IBM Support team performs the second phase, which is analyzing the Java archive (JAR) file that the collector program produces.

The collector program runs to completion as it creates the JAR file, despite any errors that it might find. Errors might include missing files or commands. The collector tool collects as much data in the JAR file as possible.

Collector tool

The collector tool gathers extensive information about your WebSphere Application Server installation and packages it in a Java archive (JAR) file that you can send to IBM Customer Support to assist in determining and analyzing your problem.

Information in the JAR file includes logs, property files, configuration files, operating system and Java data, and the absence or level of each software prerequisite. The collector program runs to completion despite any errors that it might find. Errors might include missing files or commands. The collector tool collects as much data in the JAR file as possible.

See “Running the collector tool” for the steps for running the collector program.

You can also run the collector summary option to create a lightweight version of the information in a text file and on the console. The lightweight information is useful for getting started in communicating your problem to IBM Support.

Running the collector tool:

Use the collector tool to gather information about your system and one or more profiles. The sort of information that you gather is not something that most people use. In fact, the collector tool packages its output into a JAR file. People normally use the collector tool when directed to do so by IBM Support, when gathering information to send to IBM as part of reporting a problem. IBM includes the collector tool in the product code, along with other tools that help capture the information that you must provide when reporting a problem. The collector tool is part of a strategy of making problem reporting as easy and complete as possible.

Run the collector tool from the root user or from the administrator user (Windows systems) to access system files that contain information about kernel settings, installed packages, and other vital data.

The collector tool is a Java application that requires a Java Runtime Environment (JRE) to run.

The tool is within the installation root directory for WebSphere Application Server. But you run the tool from a working directory that you create outside of the installation root directory. This procedure describes both of those steps and all of the other steps for using the tool and reporting the results from running the tool.

The tool collects information about the default profile if you do not use the optional parameter to identify another profile.

1. Log on to the system as root or a member of the administrator group on a Windows platform.
2. Make a working directory where you can start the collector program.
3. Run the **STRQSH** command from the CL command line to prepare to run the collector program.
4. Make the working directory the current directory.

   Run the following command from Qshell:

   `cd workingDirectory`

   The collector program writes its output JAR file to the current directory. The program also creates and deletes a number of temporary files in the current directory. Creating a work directory to run the collector program avoids naming collisions and makes cleanup easier. You cannot run the collector tool in a directory under the installation root directory for WebSphere Application Server.
5. Run the collector program by entering the fully qualified command from the command line of the working directory.

   Run the following command from Qshell:

   `app_server_root/bin/collector`

   Use the command with no additional parameter to gather one copy of the profile data and data from each server in the node, and to store the data in a single JAR output file.

   Use the following command to gather data from a specific profile that might not be the default profile:

   Run the following command from Qshell:

   `app_server_root/bin/collector -profileName profile_name`

   Use the following command to gather data from a specific server that might be giving you problems:

   Run the following command from Qshell:

   `app_server_root/bin/collector -servername server_name`

   Combine the two parameters to identify a particular server in a particular profile.

   Set the path to locate the proper command file in the *app_server_root*/bin directory if you do not want to issue a fully qualified command.

WebSphere Application Server

The collector program creates the Collector.log log file and an output JAR file in the current directory.

The name of the JAR file is composed of the host name, cell name, node name, and profile name:
`host_name-cell_name-node_name-profile_name.JAR`

The Collector.log log file is one of the files collected in the *host_name-cell_name-node_name-profile_name*.JAR file.

Send the *host_name-cell_name-node_name-profile_name*.JAR file to IBM Support for analysis.

Analyzing collector tool output

The first step in using the collector tool on your WebSphere Application Server product is to run the tool to produce a Java archive (JAR) file as output. The second step in using the collector tool is to analyze its output. The preferred method of performing this analysis is to send the JAR file to IBM Support for analysis. However, you can use this topic to understand the content of the JAR file if you perform your own analysis.

You can view the files contained in the JAR file without extracting the files from the JAR file. However, it is easier to extract all files and view the contents of each file individually. To extract the files, use one of the following commands:
- `jar -xvf WASenv.jar`
- `unzip WASenv.jar`

Wasenv.jar stands for the name of the JAR file that the collector tool creates.

The JAR file contains:
- A collector tool log file, `collector.log`
- Copies of stored WebSphere Application Server files and their full paths that are located under directory root in the JAR file
- Java information in a directory named `Java`
- A JAR file manifest

Tips and suggestions
- Unzip the JAR file to an empty directory for easy access to the gathered files and for simplified cleanup.
- Check the `collector.log` file for errors:
  - Some errors might be normal or expected. For example, when the collector attempts to gather files or directories that do not exist for your specific installation, it logs an error about the missing files.
  - A non-zero return code means that a command that the collector tool attempted to run does not exist. This might be expected in some cases. If this type of error occurs repeatedly, there might actually be a problem.

Collector summary
WebSphere Application Server products include an enhancement to the collector tool beginning with Version 5.0.2, known as the *collector summary option*.

The collector summary option helps you communicate with WebSphere Application Server technical staff at IBM Support. Run the collector tool with the -Summary option to produce a lightweight text file and console version of some of the information in the Java archive (JAR) file that the tool produces without the -Summary parameter. You can use the collector summary option to retrieve basic configuration and prerequisite software level information when starting a conversation with IBM Support.

The collector summary option produces version information for the WebSphere Application Server product and the operating system as well as other information. It stores the information in the Collector_Summary.txt file and writes it to the console. You can use the information to answer initial questions from IBM Support or you can send the Collector_Summary.txt file directly to IBM Support.

When running the collector tool on WebSphere Application Server for i5/OS, an additional file named WAS_Collection_timestamp.html is created in the logs directory of the profile where the collector tool runs. If you do not specify the -profileName option, the profile is the default profile. The HTML file is sent to IBM Support when you initiate a PMR.

Run the **collector** command to create the JAR file if IBM Support needs more information to solve your problem.

To run the collector summary option, start from a temporary directory outside of the WebSphere Application Server product installation root directory and enter one of the following commands:

- *app_server_root*/bin/collector -Summary [-profileName profileName]

  The -profileName option specifies the profile to summarize.

Configuring first failure data capture log file purges

The first failure data capture (FFDC) log file saves information that is generated from a processing failure. These files are deleted after a maximum number of days has passed. The captured data is saved in a log file for analyzing the problem.

The first failure data capture (FFDC) feature preserves the information that is generated from a processing failure and returns control to the affected engines. The captured data is saved in a log file for analyzing the problem. FFDC is intended primarily for use by IBM Service. FFDC instantly collects events and errors that occur during the WebSphere Application Server runtime. The information is captured as it occurs and is written to a log file that can be analyzed by an IBM Service representative. The data is uniquely identified for the servant region that produced the exception.

The FFDC configuration properties files are located in the properties directory under the WebSphere Application Server product installation. There are three properties files, but only the ffdcRun.properties file should be modified. You can set the ExceptionFileMaximumAge property to configure the amount of days between purging the FFDC log files. The value of the ExceptionFileMaximumAge property must be a positive number. The FFDC feature does not affect the performance of the WebSphere Application Server product.

Perform the following steps to configure the number of days between the FFDC log file purges. The value is in days.

1. Change the value for the *ExceptionFileMaximumAge* property to the number of days between the FFDC log file purges. The value of the *ExceptionFileMaximumAge* property must be a positive number. The default is seven days. For example, ExceptionFileMaximumAge = 3 sets the default time to three days. The FFDC log file is purged after three days.
2. Save the ffdcRun.properties file and exit.

The FFDC file management function removes the FFDC log files that have reached the maximum age and generates a message in the SystemOut.log file.

Getting IBM Support Assistant

IBM® Support Assistant helps you find and use various IBM Support resources, such as updated documentation and problem determination tools.

IBM Support Assistant is available on the WebSphere Application Server Supplements CD and on the IBM site at http://www-306.ibm.com/software/support/isa/. IBM Support Assistant offers the following components to help you with software questions:

- A search component, which helps you access pertinent Support information in multiple locations.
- A support links component, which provides a convenient location to access various Web resources such as IBM product sites, IBM support sites, and links to IBM news groups.
- A service component, which helps you submit an enhanced problem report that includes key system data to IBM. The data collector of the service component supports problem-specific data collection.
- A pluggable tools framework, with which you can download and use problem determination and other support tools.

1. Install the tool. Use the Launchpad from the WebSphere Application Server Supplements CD. If you cannot use the Launchpad, you can still install IBM Support Assistant by running the appropriate installation executable file that is located on the CD.

2. Start the tool. The IBM Support Assistant is a Web application that is displayed in the default system-configured Web browser.

To learn more about how to use IBM Support Assistant, click the **Help** link in the IBM Support Assistant window.

Use the **Updater** tab within IBM Support Assistant to check for updates and pluggable tools, such as Memory Dump Diagnostic for Java™, that might apply to your product.

Diagnosing out-of-memory errors and Java heap memory leaks

Memory Dump Diagnostic for Java is an offline tool for diagnosing root causes behind memory leaks in the Java heap. The tool analyzes common formats of memory dumps (heap dumps) from the Java virtual machine (JVM) in which the WebSphere® Application Server runs. The analysis of memory dumps is targeted toward identifying data structures within the Java heap that might be root causes of memory leaks. The analysis also identifies a summarized set of object groups that contribute significantly to the Java heap footprint of the application. The tool is capable of analyzing very large memory dumps from production-environment application servers that exhibit out-of-memory issues.

This tool works with the memory dumps that are generated from WebSphere Application Server, not from the Memory Dump Diagnostic for Java tool. The following dump formats are supported:

- IBM Portable Heap Dump (.phd), for WebSphere Application Server versions 5.1.x and later on most platforms
- IBM Text, for WebSphere Application Server versions 4.x and 5.0.x on most platforms

Memory leaks can occur in Java applications when object references are unintentionally held onto after the references are no longer needed. This problem prevents the Java garbage collection process from freeing memory, even though the Java language has a built-in garbage collection mechanism that frees the programmer from explicit object deallocation responsibilities. Memory leaks are hard to diagnose in large complex Java applications, because of the large number of objects in the Java heap and because of the complex relationships between these objects.

Two types of analysis mechanisms are available: single-dump analysis and the comparative analysis of two dumps. The tool lists suspected data structures and data types and displays contents of the memory dump in an interactive browser-based Web application. The tool shows footprint analysis results in a graphical layout of significant sets of data types that have similar ownership structures. The tool shows the contents of the memory dump in an interactive tree view for browsing and in two table views of objects and data types, respectively. The tree view enables you to look for all incoming and outgoing references for each object and to see the location of a container object within each data structure with a suspected memory leak.

Analyzing memory dumps in an offline fashion provides a low-overhead mechanism for identifying root causes behind memory leaks. This mechanism is particularly suitable for large applications that are running in production or in stress-test environments, where memory leaks are often detected first.

1. Install IBM Support Assistant on a computer that is separate from the application server installation to be analyzed. Use a non-production computer with at least 5 gigabytes (GB) of free disk space, at least 1.5 GB of RAM, and an Intel® processor that runs at 2 GHz or faster, preferably on the Linux® platform.
2. Enable verbose garbage collection on the application server installation to be analyzed.

   Verbose garbage collection information helps to rule out basic configuration issues and memory-leak issues from fragmentation or native memory leaks. For more information about how to enable verbose garbage collection on IBM platforms, see IBM developer kits: Diagnosis documentation.
3. **Optional:** Enable lightweight memory-leak detection in WebSphere Application Server.

Enabling lightweight memory-leak detection can help with early detection of abnormal memory usage behavior and for automatically triggering heap dumps.

4. Enable the JVM heap dump feature for WebSphere Application Server.

   Refer to the documentation that is supplied with the tool when you start from IBM Support Assistant.

5. When the heap dump is available, run the Memory Dump Diagnostic for Java tool.
   a. Start IBM Support Assistant.
   b. In IBM Support Assistant, select the **Tools** tab.
   c. On the left side, click **WebSphere 6.1**.
   d. On the right side, click **Memory Dump Diagnostic for Java**.

The possible root causes of memory leaks are identified.

Memory leaks in the Java heap produce java.lang.OutOfMemoryError exceptions in log files. However, not all out-of-memory errors are caused by Java heap memory leaks. Out-of-memory errors can also be caused by the following conditions:

- Java heap fragmentation. This fragmentation occurs when no contiguous chunk of free Java heap space is available from which to allocate Java objects. Various causes for this problem exist, including the presence of pinned or dosed objects or because of the repeated allocation of large objects.
- Memory leaks in native heap. This problem occurs when a native component, like DB2® connections, is leaking.

For both of these conditions, the out-of-memory error can occur in spite of large amounts of free Java heap space. Consequently, the Memory Dump Diagnostic for Java tool might not be effective in determining root causes in these cases.

Troubleshooting help from IBM

If you are not able to resolve a WebSphere Application Server problem by following the steps described in the Troubleshooting guide, by looking up error messages in the message reference, or looking for related documentation on the online help, contact IBM Technical Support.

Purchase of WebSphere Application Server entitles you to one year of telephone support under the Passport Advantage program. For details on the Passport Advantage program, visit http://www.lotus.com/services/passport.nsf/WebDocs/Passport_Advantage_Home.

The number for Passport Advantage members to call for WebSphere Application Server support is 1-800-237-5511. Please have the following information available when you call:

- Your Contract or Passport Advantage number.
- Your WebSphere Application Server version and revision level, plus any installed fixes.
- Your operating system name and version.
- Your database type and version.
- Basic topology data: how many machines are running how many application servers, and so on.
- Any error or warning messages related to your problem.

IBM Support has documents that can save you time gathering information needed to resolve this problem. Before opening a PMR, see the IBM Support page.

Tracing

WebSphere Application Server support engineers might ask you to enable tracing on a particular component of the product to diagnose a difficult problem.

Consulting

For complex issues such as high availability and integration with legacy systems, education, and help in getting started quickly with the WebSphere product family, consider using IBM consulting services. To learn about these services, browse the Web site http://www-1.ibm.com/services/fullservice.html.

Diagnosing and fixing problems: Resources for learning

In addition to the information center, there are several Web-based resources for researching and resolving problems related to the WebSphere Application Server.

The WebSphere Application Server support page

The official site for providing tools and sharing knowledge about WebSphere Application Server problems is the WebSphere Application Server support page: http://www.ibm.com/software/webservers/appserv/support.html. Among the features it provides are:

- A search field for searching the entire support site for documentation and fixes related to a specific exception, error message, or other problem. Use this search function before contacting IBM Support directly.
- *Solve a problem* links take you to specific problems and resolutions documented by WebSphere Application Server technical support personnel.
- The *Download* links provide free WebSphere Application Server maintenance upgrades and problem determination tools.

  The fix packs and refresh packs for the WebSphere Application Server for i5/OS products are also provided in the group PTF for the WebSphere Application Server product. The WebSphere Application Server group PTF also includes other i5/OS group PTFs which are necessary for running your application servers. For more information see the PTFs page.
  - *fixes* are software patches which address specific WebSphere Application Server defects. Selecting a specific defect from the list in the *Fixes by version* page takes you to a description of what problem the fix addresses.
  - Fix packs are bundles of multiple fixes, tested together and released as a maintenance upgrade to WebSphere Application Server. Refresh packs are fix packs that also contain new function. If you select a fix pack from this page, you are taken to a page describing the target platform, WebSphere Application Server prerequisite level, and other related information. Selecting the *fix list* link on that page displays a list of the fixes which the fix pack includes. If you intend to install a fix which is part of a fix pack, it is usually better to upgrade to the complete fix pack rather than to just install the individual fix.

Accessing WebSphere Application Server support page resources

Some resources on the WebSphere Application Server support page are marked with a key icon. To access these resources, you must supply a user ID and password, or register if do not already have an ID. When registering, you are asked for your contract number, which is supplied as part of a WebSphere Application Server purchase.

WebSphere Developer Domain

The Developer Domains are IBM-supported sites for enabling developers to learn about IBM software products and how to use them. They contain resources such as articles, tutorials, and links to newsgroups and user groups. You can reach the WebSphere Developer Domain at http://www7b.software.ibm.com/wsdd/.

The IBM Support page

IBM Support has documents that can save you time gathering information needed to resolve this problem. Before opening a PMR, see the Must gather documents for information to gather to send to IBM Support.

Debugging Service details

Use this page to view and modify the settings used by the Debugging Service.

To view this administrative console page, click **Servers > Application Servers** > *server name* > **Debugging Service**.

The steps below describe how to enable a debug session on WebSphere Application Server. Debugging can prove useful when your program behaves differently on the application server than on your local system.

Enable service at server startup

Specifies whether the server will attempt to start the Debug service when the server starts.

JVM debug port

Specifies the port that the Java virtual machine will listen on for debug connections.

JVM debug arguments

Specifies the debugging argument string used to start the JVM in debug mode.

Debug class filters

Specifies an array of classes to ignore during debugging. When running in step-by-step mode, the debugger will not stop in classes that match a filter entry.

Configuration problem settings

Use this page to identify and view problems that exist in the current configuration.

To view this administrative console page, click **Troubleshooting > Configuration Problems** in the console navigation tree.

To view a configuration problem, click **Configuration Validation** in the console navigation tree, then select the type of configuration you want to view.

Configuration document validation

Use these fields to specify the level of validation to perform on configuration documents.

Maximum
Selecting **Maximum: Validate all documents** turns on validation for all documents, regardless of whether or not they are extracted, and regardless of the relationships between the documents.

High Selecting **High: Validate extracted, parent, and sibling documents** turns on validation for extracted documents and their parent documents, and turns on validation for the sibling documents of the documents which have been extracted. For example, if **High** validation is selected, and if the `server.xml` document is extracted, when performing validation, validation is performed on the three documents: `server.xml`, `node.xml`, and `cell.xml`. and on the two sibling documents `variables.xml` and `resources.xml` within the server1 directory.

Medium
Selecting **Medium: Validate extracted and parent documents** turns on validation for the documents which have been extracted by the user interface, and also turns on validation of the parent documents of the documents which have been extracted. For example, using the partial directory structure from above, if **Medium** validation is selected, and if the `server.xml` document is extracted, when performing validation, validation is performed on all three of the documents `server.xml`, `node.xml`, and `cell.xml`.

Low Selecting **Low: Validate extracted documents** turns on validation for just those documents which have been extracted by the user interface.

None Selecting **None** disables validation. No configuration documents are validated.

Enable Cross validation

Enables cross validation of configuration documents. Enabling cross validation enables comparison of configuration documents for conflicting settings.

Configuration Problems

Displays current configuration problem error messages. Click a message for detailed information about the problem.

Scope

Sorts the configuration problem list by the configuration file where each error occurs. Click a message for detailed information about the problem.

Message

Displays the message returned from the validator.

Explanation

A brief explanation of the problem.

User action

Specifies the recommended action to correct the problem.

Target Object

Identifies the configuration object where the validation error occurred.

Severity

Indicates the severity of the configuration error. There are three possible values for severity.

Error This means that there is a problem with the configuration that might cause partial or complete failure of server function. This is the most severe warning.

Warning
This means that there is a problem with the configuration that might cause a failure of server function, or that might cause the server to function in an unexpected manner.

Information
A setting of the configuration that is unexpected and noteworthy, which requires customer notification. Information is used when the configuration has a value which is probably okay, but should be double checked by the administrator. This is the least crucial level of severity.

Local URI

Specifies the local URI of the configuration file where the error occurred.

Full URI

Specifies the full URI of the configuration file where the error occurred.

Validator classname

The classname of the validator reporting the problem.

Runtime events

Use the Runtime event pages of the administrative console to view the events published by application server classes.

To view these administrative console pages, click **Troubleshooting**. Expand **Runtime Messages** and click either **Runtime Error**, **Runtime Warning**, or **Runtime Information**.

Separate pages show error events, warning events, and informational events. Each page displays events in the same format.

You can adjust the number of messages that appear on the page in the **Preferences** settings.

Fields on the page include:

Timestamp
When the event occurred.

Message originator
Internal application server class that published the event.

Message
Identifier and short description of the event.

Click a message to view event details.

Message details

Use the Message Details panel of the administrative console to view detailed information about errors, warnings, and informational messages.

To view these administrative console pages, click **Troubleshooting**. Expand **Runtime Messages** and click either **Runtime Error**, **Runtime Warning**, or **Runtime Information**. Click a message to display this panel.

Each message has the following general property fields:

Message
The message ID and text.

Message type
Error, Warning, or Information.

Explanation
A description of the message.

User action
What you should do about the message.

Message originator
The name of the product class that originated the message.

Souce object type
The name of the component that originated the message.

Timestamp
The date and time that the message originated.

Thread ID
The thread identifier.

Node name
The name of the node of the application server that originated the message.

Server name
The name of the application server process that originated the message.

Diagnostic Provider ID
The Diagnostic Provider ID of the component that originated the message. Click on Configuration Data, State Data, or Tests to run the corresponding diagnostic action against the originating component. A Diagnostic Provider ID will not be supplied with all messages.

Showlog commands for Common Base Events

The showlog command converts the service log from binary format into plain text.

Purpose

These showlog commands to produce output in Common Base Event XML format.

- `showlog -format CBE-XML-1.0.1` *filename*

where:

filename
Is the service log file name.

For examples of showlog scripts, see Showlog Script.

Working with Diagnostic Providers

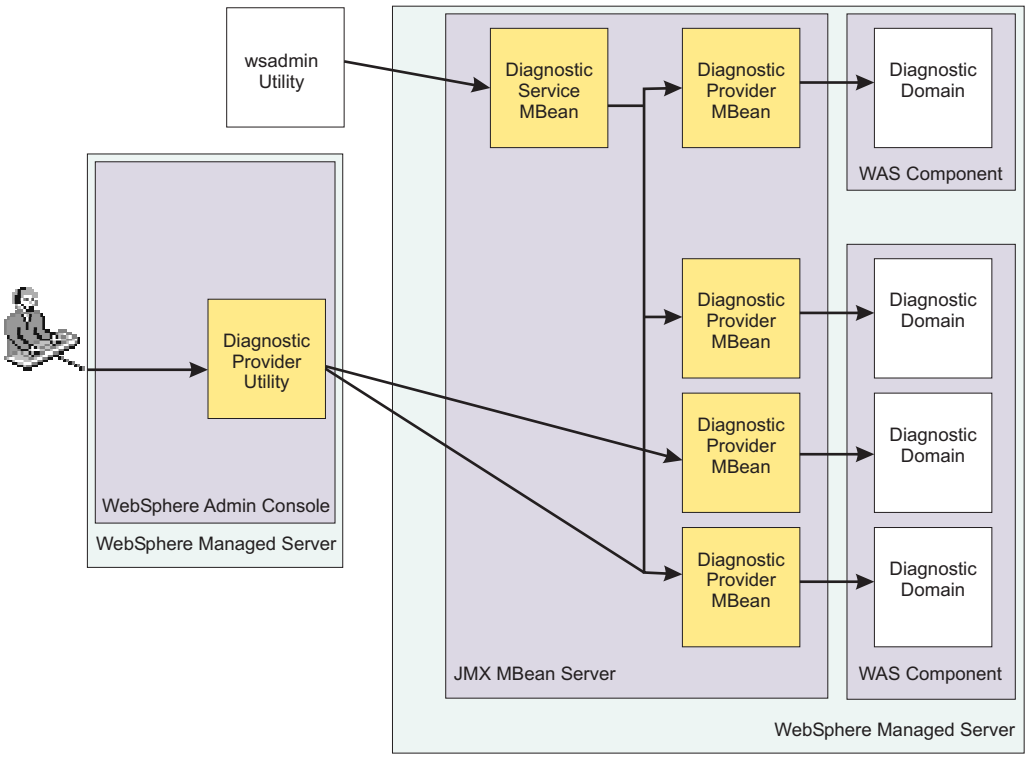
Diagnostic Providers enable you to query the startup configuration, current configuration, and current state of a diagnostic domain. In addition, Diagnostic Providers can also provide access to any self diagnostic tests that are available from a diagnostic domain.

WebSphere Application Server provides a set of facilities to accomplish this. The Diagnostic Provider Utility is a simple front end in the administration console that presents the available set of Diagnostic Providers and enables you to work with them.

Learn about Diagnostic Providers

Diagnostic Providers

WebSphere Application Server components can be considered as being divisible into *diagnostic domains*. A diagnostic domain refers to a set of classes within the component that share a set of diagnostics. Some larger components might have multiple diagnostic domains. For example, the Connection Manager logically consists of multiple data sources and connection factories that each have separate diagnostic domains



This image shows the relationships between the parts that make up the Diagnostic Provider (DP) utility.

Diagnostic Provider MBeans

A single diagnostic domain receives its diagnostic services from a Diagnostic Provider MBean. The Diagnostic Provider MBean enables you to query the startup configuration, current configuration, and current state of the diagnostic domain. In addition, Diagnostic Provider MBeans can also provide access to any self diagnostic tests that are available from the diagnostic domain. Some characteristics of Diagnostic Provider MBeans include:

- Diagnostic Provider MBeans are Java Management Extensions (JMX) MBeans
- Diagnostic Provider MBeans all implement a DiagnosticProvider interface which includes methods for configuration dumps, state dumps, and self diagnostic tests
- Diagnostic Provider MBeans provide a way to expose information about running components so administrators can more easily debug problems related to those components. As with other MBeans running in WebSphere Application Server, they can be accessed from JMX client code, or through the *wsadmin* tool.

Diagnostic Provider Infrastructure

Diagnostic Provider MBeans are efficient at delivering Java object representations of configuration, state, and self test information. This is good for when programs interact. For human users to access the information, WebSphere Application Server provides a set of facilities to extend the value of Diagnostic Provider MBeans.

The Diagnostic Service MBean
: provides methods to convert Diagnostic Provider MBean output into human readable formats. The Diagnostic Service MBean also provides some methods to facilitate looking up the Diagnostic Provider MBeans on the same server as the Diagnostic Service MBean. For administrators that want to access diagnostic data from a command line, the wsadmin tool can be used directly with the Diagnostic Service MBean to get formatted results

The Diagnostic Provider utility
a set of panels included in the WebSphere Application Server administration console through which administrators can interact with Diagnostic Provider MBeans. The Diagnostic Provider utility is a simple front end in the administration console that presents the available set of Diagnostic Provider MBeans present on each managed server, and provides a means to execute and view the results of configuration dumps, state dumps, and diagnostic self tests.

.

The purpose of Diagnostic Providers

Diagnostic Providers give you more information for quickly discovering and diagnosing system problems. The following scenario contrasts the experience of a n administrator working with a component that does not have a Diagnostic Provider to one that does.

When the administrator works with a component that is without a Diagnostic Provider, the events are as follows:
1. A log entry indicates that a particular component is experiencing a problem.
2. The system administrator sees the log entry through the runtime messages panel.
3. The system administrator cannot tell what is wrong, so calls IBM support for assistance, with a potentially ill-defined problem.

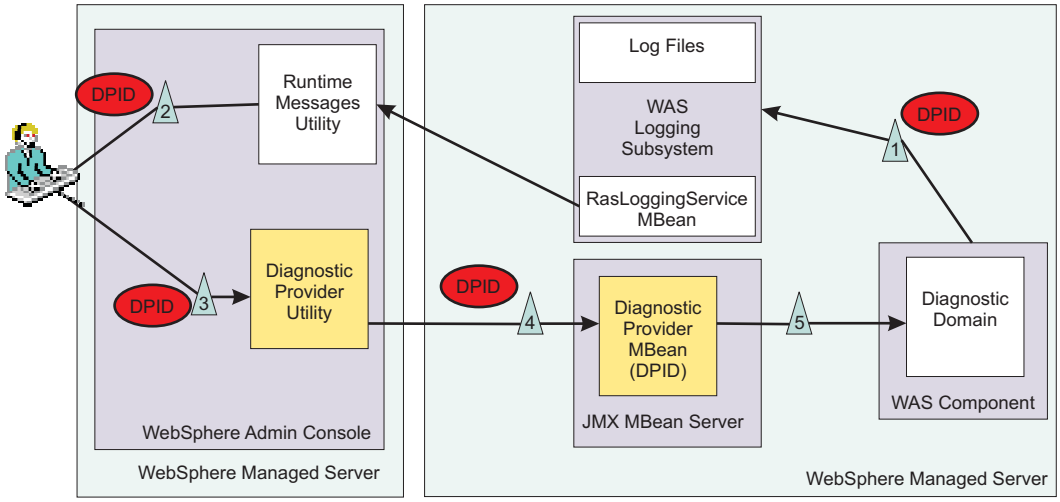
When the administrator works with a component with a Diagnostic Provider, and the Diagnostic Provider ID is registered with the component's logger, the situation changes as follows:
1. A log entry that contains a Diagnostic Provider ID (DPID) indicates that something has gone wrong in a specific component.
2. The system administrator sees the log entry through the runtime messages panel.
3. The administrator clicks a button on the runtime message panel to execute a state dump or a configuration dump, or to be taken to the list of component self tests.
4. From the self test, the administrator is warned that the component is configured in a way that could lead to poor performance or failures.

Furthermore, when the administrator works with a component with a Diagnostic Provider, and the Diagnostic Provider ID is **not** registered with the component's logger, the situation might unfold like this:
1. A log entry which doesn't contain a DPID indicates that something has gone wrong in a component.
2. The system administrator sees the log entry through the runtime messages panel.
3. The system administrator uses the administrative console to navigate through the available set of Diagnostic Providers and selects one that sounds appropriate.
4. He runs a configuration dump, a state dump, or a self diagnostic test against the Diagnostic Provider to collect information about the component.
5. From the state dump, the administrator is able to notice that the component state is not what would be expected for its workload.
6. The administrator works with the test team to determine which of the flows is causing the state of the component to diverge from what is expected (as evidenced by repeated execution of the state dump).

Diagnostic Provider IDs

A Diagnostic Provider ID (DPID) is the unique address of a Diagnostic Provider MBean. Components that have associated Diagnostic Provider MBeans can include the DPID in their log entries. Diagnostic Provider IDs are implemented in WebSphere Application Server as Java Management Extensions (JMX) *MBean ObjectNames*, and can be used at JMX MBean servers to look up Diagnostic Provider MBeans.

By including the String representation of the DPID in each logged message, the message can be tracked back to the Diagnostic Provider related to the component. A method is provided to associate Diagnostic Provider IDs with Loggers (from the *java.util.logging* logging API).



The diagram above shows how the use of DPIDs in log entries enables callbacks to the component that originally created the log entry.

1. Shows the component logging with a DPID included in the log entry.
2. The administrator examines the log entry through the Runtime Messages Utility and notices that the entry has a link to a Diagnostic Provider.
3. The administrator uses the link to gain access to the relevant MBean in the Diagnostic Provider Utility .
4. The Diagnostic Provider Utility contacts the Diagnostic Provider MBean to ask for more information.
5. The request for more information is sent back to the source of the original log entry.
6. The response from the Diagnostic Provider is provided in the administration console.

Diagnostic Provider configuration dumps, state dumps, and self tests

The Diagnostic Provider (DP) infrastructure allows for a software component or stack product in the WebSphere Application Server space to expose key information about its configuration, current state, and current ability to perform operations. The methods that expose this information might be driven as a result of a message put out by the component (by a logger which automatically includes the Diagnostic Provider ID in each message), or might be driven as a result of an overall system health-check when an administrator or automated tool is monitoring the system.

Configuration dumps

A *Configuration dump* is an operation you can perform on a Diagnostic Provider to list the startup or current values of the configuration attributes for the DP. The name for each data item in this dump should reflect its disposition. That is, each item should be called *startup-xxx* or *current-xxx* to show whether this is a startup or current value. The collection of attributes returned from this operation can be thought of as the **payload** of the configuration dump. More information about payloads can be found in “Diagnostic Provider method implementation” on page 140.

You can find several ways to filter the output of a configuration dump in “Diagnostic Provider registered attributes and registered tests” on page 135.

State dumps

A *State dump* is similar to a configuration dump, but it differs in two key areas. First, a state dump displays current information about the operation of a component. An example is a connection pool. A configuration dump can show *DataSource name*, the *minConnections* (configured or current), the *maxConnections*, the *DataBase name*, and so on. A state dump is more likely to show the current connections in use, the high concurrent use count, the number of times the pool has been expanded, the average time between requesting a connection and returning it, and so forth.

State dumps can be impacted by the values in the State Collection Specification. This is a dynamic specification that controls additional data collection that the component can do at runtime. If additional data is being collected, then a State dump might display more information. The same filters and payload information that apply to Configuration dumps (see “Diagnostic Provider registered attributes and registered tests”) apply to State dumps.

Self Diagnostic tests

Self diagnostic tests are non-invasive operations that a Diagnostic Provider exposes. *Non-invasive* means that if they modify anything for the test, the conclusion of the test reverses the modification. These tests give an administrator the option to test simple functions of a component to see if it is able to perform them.

The filters for a self diagnostic test apply to the test itself, not to the output of the test. A typical use of Self Diagnostic tests could be for a pool manager of some sort to pull an object out of the pool and return it to the pool to verify that this operation can still be performed, and with acceptable performance.

Diagnostic Provider registered attributes and registered tests

Each Diagnostic Provider (DP) provides a list of state dump attributes, configuration dump attributes, self tests, and self test attributes. The tests are operations that the DP can perform. The attributes are pieces of information that are available for collection from a Configuration dump, a State dump, or a specific Self Diagnostic test.

Each attribute can be seen as a piece of information with a label on it. Each attribute is also considered to be either *registered* or *not registered*. A registered attribute is one that should be available from one release of WebSphere Application Server to the next. A nonregistered attribute might not be available in its current form in future releases of the product (no commitment has been made).

When performing a Configuration dump, a State dump, or a Self Diagnostic test, an administrator or automatic tooling can request *only registered* values, or *all* values, depending on the needs of the administrator or tool. Note that the option of filtering results is only available through the Diagnostic Provider’s Java Management Extension (JMX) MBean interface, which you can access programmatically or through the wsadmin tool.

Diagnostic Provider registered attributes and registered tests detail: This article includes a discussion of the DiagnosticProviderRegistration Extensible Markup Language (XML) file and how it is used in conjunction with the method signatures to filter the results of calling the various methods.

The DiagnosticProviderRegistration XML file

This XML defines the configuration information, state information, and self diagnostic tests exposed by the component. In the configuration and state information, the key working unit is referred to as the attribute. Specification of an attribute is as follows:

```
<attribute>
        <id><Regular Expression representing the attribute name></id>
        <descriptionKey><MsgKey into a ResourceBundle for localization of the label></descriptionKey>
        <registered>true</registered>
   </attribute>
```

The parts are as follows:

ID: The attribute’s name. This name can be expressed with wildcard characters conforming to regular expression syntax. The registered attribute ID is used in the following places:

- Within Diagnostic Provider configuration dump and state dump methods to determine which attributes to return.
- In the administration console to match description keys to attributes returned from a request to a Diagnostic Provider for a configuration dump, state dump, or self diagnostic operation.

As an example, if a configuration dump returns an attribute with ID **cachedServlet-MyServlet-servletPath** to the administration console, the administration console could use the *descriptionKey* corresponding to the attribute registered as **<id>cachedServlet-.*-servletPath</id>** when selecting what description text to put next to this attribute's name and return values.

descriptionKey:
This is a key into a *resourceBundle* for localization.

registered:
This is a boolean qualifying whether this attribute will be available from one release of the software to the next. If registered is **true**, then this attribute should be available in the next release. If registered is **false**, then there is no guarantee that this attribute will continue to exist. Automation should use some caution when handling non-registered attributes.

Specification of a selfDiagnosticTest is as follows:

```
<test>
       <id><Regular Expression for the name of the test></id>
       <descriptionKey><MsgKey into a ResourceBundle for localization of the label></descriptionKey>
           <attribute><One or more attributes which will be output from this test></attribute>
</test>
```

The parts are as follows:

ID: Similar to the ID for the attribute, but in this case, describing the test to be performed instead of the attribute to be returned.

descriptionKey:
This is a key into a resourceBundle for localization.

Method interfaces

```
public DiagnosticEvent [] configDump(String aAttributeId, boolean aRegisteredOnly);
public DiagnosticEvent [] stateDump(String aAttributeId, boolean aRegisteredOnly);
```

These methods invoke the configuration or state dump on the component, and specify a regular expression filter for the attributes to return as well as filtering the output to include all matching attributes, or only those attributes which are registered. This enables the administrator or automated software driving the method to specify a subset of the overall fields (especially important if many attributes are exposed or if the State Collection Specification increases the amount of data available). The following helper methods are available to assist with filtering the output.

To take a list of Attributes that are available to return, and filter them:

```
    public static AttributeInfo [] queryMatchingDPInfoAttributes(String aAttributeId,
        AttributeInfo [] inAttrs, String [] namesToCheck, boolean aRegisteredOnly) {
```

To take a single Attribute that is available to return, and filter it:

```
    public static AttributeInfo queryMatchingDPInfoAttributes(String aAttributeId,
        AttributeInfo [] inAttrs, String nameToCheck, boolean aRegisteredOnly) {
```

To go through a populated set of Attribute Information and remove unneeded parts:

```
    public static void filterEventPayload(String aAttributeId, HashMap payLoad) {
```

For details on these messages, please review the API documentation for the **DiagnosticProviderHelper** class. The basic concept is that, once the component knows what attributes are able to be returned, the helper method will determine which of them should be returned based on the regular expression logic and registration boolean.

The selfDiagnostic Method interface here is similar to that of Configdump and Statedump:

```
public DiagnosticEvent[] selfDiagnostic(String aTestId, boolean aRegisteredOnly)
```

The difference is that the first parameter is a regular expression filter for which test to run.

Diagnostic Provider names

In addition to the Diagnostic Provider ID (DPID), each component that implements the Diagnostic Provider interface must have a Diagnostic Provider Name. While the DPID must be unique within the entire WebSphere Application Server domain, the Diagnostic Provider Name need only be unique within the Java Virtual Machine (JVM).

Unlike the Diagnostic Provider ID, which tends to be long and not human-friendly, the Diagnostic Provider Name should be shorter and easier to read. In addition, by convention it should end in *DP*. The Diagnostic Service MBean (see "The simpler interfaces provided by the Diagnostic Service MBean") can drive methods on a Diagnostic Provider using its name.

The simpler interfaces provided by the Diagnostic Service MBean

All services for a Diagnostic Provider (DP) are also available through a Java Management Extensions (JMX) interface known as the *Diagnostic Service* interface. The Diagnostic Service interface enables administrators to drive methods against DPs using the Diagnostic Provider Name or Diagnostic Provider ID. When formatted output is requested of the Diagnostic Service, it is localized to the client Locale. This makes the Diagnostic Service MBean ideal for clients using an interface where consuming complex Java objects, such as those returned from the Diagnostic Provider MBeans, is not feasible. An example of such an interface is the wsadmin tool.

The simpler interfaces provided by the Diagnostic Service MBean detail: The Diagnostic Service interface provides four signatures for each of the key methods available on the Diagnostic Providers (*configDump*, *stateDump*, and *selfDiagnostic*) objects. Because these method signatures look so similar, this example shows it all through the configDump methods. The four Diagnostic Service methods that map to configDump on a Diagnostic Provider are:

```
public DiagnosticEvent [] configDump(String aDPName, String aAttributeIdSpec, boolean aRegisteredOnly)
public DiagnosticEvent [] configDumpById(String aDPid, String aAttributeIdSpec, boolean aRegisteredOnly)
public String [] configDumpFormatted(String aDPName, String aAttributeIdSpec,
       boolean aRegisteredOnly, Locale aLocale)
public String [] configDumpFormattedById(String aDPid, String aAttributeIdSpec,
       boolean aRegisteredOnly, Locale aLocale) {
```

The first two return exactly what the Diagnostic Provider does. The second two methods act as a pass-through to the actual Diagnostic Provider, but they take the array of Diagnostic Events that the Diagnostic Provider returns, and convert it into a more easily consumable *String* array. In addition, these methods handle localizing the output to the appropriate Locale. Important to note is that the same method can be driven using the Diagnostic Provider ID, or the Diagnostic Provider Name.

Creating a Diagnostic Provider

To complete this task you must have programming knowledge of your system and the proper authorities to perform the following steps.

The steps that follow outline a general process for creating Diagnostic Providers (DP).

1. Determine your diagnostic domain. Look for *configuration* MBeans that control a similar domain in the same component. Extending an existing configuration MBean with a DP interface avoids proliferation of new MBeans and has the benefit that mapping from a diagnostic MBean to a configuration MBean requires no additional information.
2. Determine what configuration attributes you want to expose. Include information that is used to configure your component from the configuration MBeans.
3. Determine what state attributes you want to expose. Anything you might want to know about the state of your component for troubleshooting can go here.
4. Determine what self diagnostic tests you will expose.
5. Determine what test attributes you will return for each self diagnostic.

6. Create your DP registration Extensible Markup Language (XML) file.
7. Create your DP implementation.
   a. To see an example, refer to “Implementing a Diagnostic Provider” on page 139 and keep in mind that most things that a Diagnostic Provider should do are already done for you in the *DiagnosticProviderHelper* class.
   b. To ensure that you do not collect unwanted data, add hooks in your component code where you need to collect state data using the *DiagnosticConfig* object.
   c. Add hooks in your component code where you need to store or be able to access configuration data.
8. Add code to register your DP implementation. Typically, the best place to do this is where your component is initialized.
9. Add Diagnostic Provider IDs (DPID) to your logged messages. Registering a DPID with a logger makes that information available in any messages logged with this logger. This enables fast paths in the DP utility to function on this particular Diagnostic Provider.
   a. Register your DPID with your loggers (for any of your loggers that you only want to associate a single DPID with).
   b. When you use multiple DPIDs with the same Logger, you can (instead of registering a single DPID with a Logger) add DPIDs to individual logging calls in the **parm[0]** position. Do not put **{0}** in the corresponding localized messages. It is bad practice to print the DPID in your messages as this would be inconsistent with messages from loggers with statically assigned DPIDs.

Conventions for Diagnostic Provider Extensible Markup Language

Some conventions to follow for Diagnostic Provider (DP) Extensible Markup Language (XML) declarations.

These guidelines are to help keep your use of Diagnostic Providers (DP) consistent.

- Include the Document type definition (DTD) for your Diagnostic Provider at the top of every DP declaration Extensible Markup Language (XML) file.
- Start all names and name segments with lower case. Use *camel case* for attribute names. That is, capitalize every initial letter in the name, except the first. For example, *traceCollectionSpec*.
- Indicate hierarchy with dashes. Dashes work better than dots because attribute names are regular expressions. For example, *traceService-traceCollectionSpec*.
- Indicate string dynamic parts to attribute names using an asterisk (*). For example,

  ```
  vhosts-.*-webgroups-.*-webapps-.*-listeners-filterInvocationListeners
  ```

  which would match vhosts-**someHost**-webgroups-**someGroup**-webapps-**someApp**-listeners-filterInvocationListeners
- Indicate numeric dynamic parts to attribute names using **[0-9]***. For example,

  ```
  vhosts-index-[0-9]*
  ```

  which would match webcontainer-vhosts-index-123
- If you have a general purpose self diagnostic test that can be run without significant performance cost, name it *general*.

Some tips for configDump implementation

- configDump should contain information used to define the component’s environment. Some examples are:
  – configuration data set by Java Management Extensions (JMX)
  – configuration from system properties, xml files, and property files
  – configuration information hard-wired and unchanging in code (such as, if a resource adapter implements interface X, or has some static final field Y, then those could indicate aspects of configuration and be included in the configDump).

- configDump should not contain dynamically registered attributes, such as:
  - a list of registered loggers (this belongs in stateDump)
  - a list of servlets in an application (this belongs in stateDump).
- configDump should be separated into 2 sections -- *startup* and *current*.
  - All configDump attributes must start with either **startup-** or **current-**.
  - The *startup* section details the component's environment at startup time. Startup configDump attributes start with **startup-** .
  - The *current* section details the component's environment at the moment the configDump is requested. Current configDump attributes start with **current-**.

Best practices for configDump

- Group related attributes using an attribute hierarchy (such as, for two attributes about the traceLog: startup-traceLog-rolloverSize=20, startup-traceLog-maxNumberOfBackupFiles=1)
- For information in the current attribute list that refers to the same thing as a startup attribute, the names of both current and startup attributes should match.
- If an attribute has no use after startup, only show it in the startup section (for example, a configuration attribute that contains a file name from which startup data is read).

Tips for choosing a Diagnostic Provider name

To ensure consistency when choosing Diagnostic Provider names to use with your components, you should consider the guidelines that follow.

Diagnostic Provider name guidelines:

- Names must be unique within a Java Virtual Machine (JVM). One Diagnostic Provider name goes uniquely with one Diagnostic Provider ID within a server.
- If necessary, names can contain a dynamic element to help with uniqueness. Of course, the dynamic element should have meaning to the administrator.
- Although not a hard limit, the static part of names should be 16 characters or less.
- The static part of names must follow the class name convention. Start with a capital letter, no spaces, and capitalize each word in the name.
- The static part of names must end with **DP**.
- Valid names contain a static part only, or a static part followed by a dash (-), followed by a dynamic part. Some valid examples:
  - ConnMgrDP-instance_specific_stuff
  - WebContainerDP
  - AdvisorDP
  - NodeAgentDP

Implementing a Diagnostic Provider

This task presumes that you have a programming knowledge of the creation of MBeans. For more information about the interaction of MBeans with WebSphere Application Server, refer to topic, Creating and registering standard, dynamic, and open custom MBeans in the *Administering applications and their environment* PDF book.

The steps that follow outline a general process for implementing a Diagnostic Provider (DP).

1. Modify the MBean descriptor Extensible Markup Language (XML). To implement a Diagnostic Provider, you must have an MBean, and the MBean should include this statement in its descriptor XML as a direct child of the MBean element:

   ```
   <parentType type="DiagnosticProvider"/>
   ```

This defines the operations, attributes, and aggregators necessary for an MBean to be a Diagnostic Provider. If you do not need to have this DP exist in z/OS Controllers, then this XML inclusion handles all z/OS specifics for your MBean.

2. Modify the MBean Implementation. Your MBean should already have a class which instantiates it and registers it with the Java Management Extensions (JMX) server.

   The first difference here is that you must define a property in the *Properties* class that is passed to the registration (and becomes part of the *ObjectName*). The property is **diagnosticProvider=true** and it can be added with a line of code such as:

   ```
   MyProps.setProperty(DiagnosticProvider.DIAGNOSTIC_PROVIDER_KEY, DiagnosticProvider.DIAGNOSTIC_PROVIDER_VALUE) ;
   ```

   The second difference is that this class should register this Diagnostic Provider with the Diagnostic Service. A helper method is available to do this:

   ```
   DiagnosticProviderHelper.registerMBeanWithDiagnosticService(DiagnosticProviderPName, DiagnosticProviderId) ;
   ```

   Obviously this must be done after the registration when the *ObjectName* can be retrieved into the *DiagnosticProviderId* string.

3. Implement the Diagnostic Provider methods.

Diagnostic Provider method implementation: To create a Diagnostic Provider (DP), you must have an MBean, and your MBean should include the following methods in its deployment Extensible Markup Language (XML) file. This can be accomplished by adding the *parentType* directive to your existing XML file (see “Implementing a Diagnostic Provider” on page 139), or by including the operations directly into your deployment XML file. The definitions needed are included in “Diagnostic Provider registered attributes and registered tests” on page 135. This defines the operations, attributes, and aggregators necessary for an MBean to be a Diagnostic Provider. The next step is for the MBean to actually implement these methods. The methods to implement include:

- “getRegisteredDiagnostics”
- “getDiagnosticProviderName” on page 141
- “getDiagnosticProviderID” on page 141
- “configDump” on page 141
- “stateDump” on page 141
- “selfDiagnostic” on page 142
- “localize” on page 142

getRegisteredDiagnostics

This method exposes the registration information for this Diagnostic Provider. It is commonly used by the DP Utility in the administration console to gather information about Diagnostic Providers that are to be displayed in the console. This method returns a **DiagnosticProviderInfo** object that is usually attained by passing the appropriate XML to a **DiagnosticProviderHelper** helper class. Here is an example:

```
public DiagnosticProviderInfo getRegisteredDiagnostics() {
  InputStream regIS= Thread.currentThread().getContextClassLoader().getResourceAsStream(
   "com/ibm/ws/xxx/SampleDP2DiagnosticProvider.xml");
  dpInfo = DiagnosticProviderHelper.loadRegistry(regIS, sDPName) ;

         if (dpInfo == null) {
            sSampleDP2MBeanLogger.logp(Level.WARNING, sThisClass, "getRegisteredDiagnostics",
             "RasDiag.DPInfo.NoGotz") ;
  }
    return dpInfo ;
 }
```

Notice that the XML is packaged and available in the *classpath* of the current *classloader*. The “Registration XML” on page 143 contains crucial information that the Diagnostic Provider uses to “Populate the payload” on page 143 and “localize” on page 142 results.

getDiagnosticProviderName

This is usually a pretty simple return of a constant as the following example shows

```
public String getDiagnosticProviderName() {
  return sDPName;
 }
```

getDiagnosticProviderID

This is usually a pretty simple return of a Java Management Extensions (JMX) object ID that MBeans can pull out of the base class method. For example:

```
public String getDiagnosticProviderId() {
  return getObjectName().toString() ;
 }
```

configDump

The *configDump* method enables the Diagnostic Provider to expose the configuration data that was in place when this Diagnostic Provider started (or the current values of them). The **DiagnosticEvent** objects that this method returns include a “Payload” on page 142 that contains the core data. The following is an excerpt from a configDump method:

```
public DiagnosticEvent [] configDump(String aAttributeIdSpec, boolean aRegisteredOnly) {
    HashMap cdHash = new HashMap(64) ;

  //
```

“Populate the payload” on page 143 DiagnosticEvent [] diagnosticEvent = new DiagnosticEvent[1] ; diagnosticEvent[0] = DiagnosticEventFactory.createConfigDump(getObjectName().toString(), ″ThisClassName″, ″configDump″, cdHash) ; return diagnosticEvent ; }

This returns an array of **DiagnosticEvent** objects. Normally, *configDump* and *stateDump* return only one object. However, the method accepts an array because on z/OS systems a server can have multiple servants, and aggregation of the output from the servants is stored in the array.

stateDump

The *stateDump* method enables the Diagnostic Provider to expose the current state data, or data about the current operating conditions of the Diagnostic Provider. The data made available can be anything likely to assist a customer, an IBM support person, or automated tooling in analyzing the health of the component and problem determination if there is an issue. The amount of data available is impacted by the State Collection Specification in effect at the time. If the current State Collection Specification involves the collection of additional data by the Diagnostic Provider, then this additional data can be exposed in the stateDump. The **DiagnosticEvent** objects that this method returns include a “Payload” on page 142 that contains the core data. The following is an excerpt from a stateDump method:

```
public DiagnosticEvent [] stateDump(String aAttributeIdSpec, boolean aRegisteredOnly) {
    HashMap sdHash = new HashMap(64) ;

  //
```

“Populate the payload” on page 143 DiagnosticEvent [] diagnosticEvent = new DiagnosticEvent[1] ; diagnosticEvent[0] = DiagnosticEventFactory.createStateDump(getObjectName().toString(), ″ThisClassName″, ″stateDump″, sdHash) ; return diagnosticEvent ; }

This returns an array of **DiagnosticEvent** objects. Normally, *configDump* and *stateDump* return only one object. However, the method accepts an array because on z/OS systems a server can have multiple servants, and aggregation of the output from the servants is stored in the array.

selfDiagnostic

The *selfDiagnostic* method enables the Diagnostic Provider to perform certain predefined activities to test key functionalities of your system. These tests should not have a lasting effect on the system. For example, if the test is to create a TCP/IP connection to a remote host, the test should also break that connection before returning its results so that the state of the component is unchanged by the test. The information returned by the test is determined by the attributes included in the test section of the XML file. The following is an excerpt from a selfDiagnostic method:

```
public DiagnosticEvent [] selfDiagnostic(String aAttributeIdSpec, boolean aRegisteredOnly) {
  TestInfo [] testInfo = dpInfo.selfDiagnosticInfo.testInfo ; // Retrieve the test registry information
  Pattern testChecker = Pattern.compile(aAttributeIdSpec) ; // Compile test regexp parm for faster checking
  ArrayList deList = new ArrayList(8) ;    // Allocate expandable list of DiagnosticEvents
  for (int i = 0; i < testInfo.length; i++) {
   if (testChecker.matcher(testInfo[i].id).matches()) {
      HashMap deHash = new HashMap(32) ;

    //
```

“Populate the payload” on page 143

```
deList.add(DiagnosticEventFactory.createDiagnosticEvent(getObjectName().toString(),
DiagnosticEvent.EVENT_TYPE_SELF_DIAGNOSTIC, DiagnosticEvent.LEVEL_INFO, ″ThisClassName″,
″selfDiagnostic″, dpInfo.resourceBundleName, ″RasDiag.SDP2.createDE3″, // MsgKey for localization //
Parms to incorporate in msg new Object [] { ″OneParm″, ″TwoParm″, ″RedParm″, ″BlueParm″}, deHash)) ;
} } DiagnosticEvent [] diagnosticEvent = new DiagnosticEvent[deList.size()] ; diagnosticEvent =
(DiagnosticEvent [])deList.toArray(diagnosticEvent) ; return diagnosticEvent ; }
```

This returns an array of **DiagnosticEvent** objects. In this example, one **DiagnosticEvent** was created from each test that matched the parameter regular expression. The Diagnostic Provider is not required to produce only one per test. The generation of “Payload” is similar to that of *configDump* and *stateDump*. Aggregation on multiple z/OS servants for an individual server concatenates the arrays from each servant.

localize

The **DiagnosticEvents** that methods return contain payload **HashMaps** that contain **MessageKeys** and **ResourceBundles**. The final consumer of these events is often not on the server, and thus may not have the appropriate *classpath* to resolve this. For this purpose, a callback to the Diagnostic Provider to localize the variables is done. A helper method, however, makes it a simple method to write, as this example demonstrates:

```
public String [] localize(String [] aKeys, Locale aLocale) {
  return DiagnosticProviderHelper.localize(dpInfo.resourceBundleName, aKeys, aLocale) ;
 }
```

Note that the **dpInfo** (DiagnosticProviderInfo) object is needed as this object includes a reference to the **ResourceBundle**.

Payload

A recurring theme in these methods is the ability to include a payload in return objects. This is a set of *name=value* pairs that include the information being exposed by the method. Diagnostic Events returned from a configDump, stateDump, or selfDiagnostic test are relatively complex Java objects. The majority of the information that is returned is contained in the **DiagnosticData** portion of the **DiagnosticEvent** object. Each attribute returned by the Diagnostic Provider is stored in an entry in a **HashMap**. There can be cascading HashMaps within a single DiagnosticEvent object (if breaking the data down into subGroups makes sense). Each HashMap entry contains either a reference to a child HashMap, or a **DiagnosticTypedValue** (which contains the value, the type of data, and a **MsgKey** for localization of the label or /name). The values to be returned should be filtered with:

- The type of method (that is, configDump, stateDump, or selfDiagnostic)

- The **AttributeIdSpec** sent in to filter the values
- The current State Collection Specification (which can impact the amount of data available).

Populate the payload

The API documentation for *DiagnosticProviderHelper.queryMatchingDPInfoAttributes* explains how to do the filtering before retrieving the data. In some cases, it is easier and helps performance for a Diagnostic Provider to retrieve all data into the Payload and then filter the HashMap after the fact. The post-population filtering can be done with the method *DiagnosticProviderHelper.filterEventPayload*. For information on use of the JavaBean type approach, see the API documentation for the *AttributeBeanInfo.populateMap* method.

Registration XML

Registration XML enables much of the information needed by the Diagnostic Provider to be externalized. It also provides a means of commonizing localization and consumption of the tests (thus aiding automation). An excerpt of this XML from a sample Diagnostic Provider follows:

```
<!DOCTYPE diagnosticProvider PUBLIC "RasDiag" "/DiagnosticProvider.dtd">

<diagnosticProvider>
    <resourceBundleName> com.ibm.ws.rasdiag.resources.RasDiagSample</resourceBundleName>
    <state>
 <attribute>
  <id>Leg-Foot</id>
  <descriptionKey>SampleDiagnostic.LegFoot.descriptionKey</descriptionKey>
  <registered>true</registered>
 </attribute>
 <attribute>
  <id>Leg-Ankle</id>
  <descriptionKey>SampleDiagnostic.LegAnkle.descriptionKey</descriptionKey>
  <registered>true</registered>
        </attribute>
    </state>
    <config>
        <attribute>
  <id>Arm-Hand-Size</id>
  <descriptionKey>SampleDiagnostic.HandSize.descriptionKey</descriptionKey>
  <registered>true</registered>
        </attribute>
        <attribute>
  <id>Leg-Foot-Size</id>
  <descriptionKey>SampleDiagnostic.FootSize.descriptionKey</descriptionKey>
  <registered>true</registered>
        </attribute>
    </config>
    <selfDiagnostic>
        <test>
  <id>Kick</id>
  <descriptionKey>SampleDiagnostic.Kick.descriptionKey</descriptionKey>
  <attribute>
   <id>Kick-Pain</id>
   <descriptionKey>SampleDiagnostic.KickPain.descriptionKey</descriptionKey>
  </attribute>
  <attribute>
   <id>Kick-Length</id>
   <descriptionKey>SampleDiagnostic.KickLength.descriptionKey</descriptionKey>
  </attribute>
        </test>
 <test>
  <id>Throw</id>
  <descriptionKey>SampleDiagnostic.Throw.descriptionKey</descriptionKey>
         <attribute>
   <id>Throw-Pain</id>
```

```
   <descriptionKey>SampleDiagnostic.ThrowPain.descriptionKey</descriptionKey>
   <registered>true</registered>
  </attribute>
  <attribute>
   <id>Throw-Length</id>
   <descriptionKey>SampleDiagnostic.ThrowLength.descriptionKey</descriptionKey>
   <registered>true</registered>
  </attribute>
        </test>
    </selfDiagnostic>
</diagnosticProvider>
```

For understanding the storage of this information into a **DiagnosticProviderInfo** object, see the API documentation for *DiagnosticProviderInfo*. For conceptual information about the purpose of the registration XML, see “Diagnostic Provider registered attributes and registered tests” on page 135.

Diagnostic Provider XML example:

Here is an example of the Diagnostic Provider Extensible Markup Language (XML).

```
version="6.0"
platform="common"
aggregationHandlerClass="com.ibm.ws.management.component.DiagnosticProviderAggregator"
description="DiagnosticProvider portion of Mbean for inclusion into MBeans implementing this interface">
  <attribute
    description="DiagnosticProviderName (not dependent on runtime, but subset of ObjectName"
    getMethod="getDiagnosticProviderName" name="diagnosticProviderName"
    type="java.lang.String" proxyInvokeType="unicall" proxySetterInvokeType="multicall"/>
  <operation
    description="Get the DiagnosticProvider ID"
    impact="INFO" name="getDiagnosticProviderId" role="operation"
    targetObjectType="objectReference" type="java.lang.String" proxyInvokeType="unicall">
    <signature/>
  </operation>
  <operation
    description="Return the registry information based on type (config/state/selfDiag)."
    impact="INFO" name="getRegisteredDiagnostics" role="operation"
    targetObjectType="objectReference"
    type="com.ibm.wsspi.rasdiag.diagnosticProviderRegistration.DiagnosticProviderInfo"
    proxyInvokeType="unicall">
    <signature/>
  </operation>
  <operation
    description="Dump the configuration information associated with managed resource."
    impact="INFO" name="configDump" role="operation"
    targetObjectType="objectReference" type="[Lcom.ibm.wsspi.rasdiag.DiagnosticEvent;"
    proxyInvokeType="multicall">
    <signature>
      <parameter description="Attribute ID to use"
        name="attributeId" type="java.lang.String"/>
      <parameter description="Report on just registered info, or all info"
        name="registeredOnly" type="boolean"/>
    </signature>
  </operation>
  <operation
    description="Dump state information for the managed resource."
    impact="INFO" name="stateDump" role="operation"
    targetObjectType="objectReference" type="[Lcom.ibm.wsspi.rasdiag.DiagnosticEvent;"
        proxyInvokeType="multicall">
    <signature>
      <parameter description="Attribute ID to use"
        name="attributeId" type="java.lang.String"/>
      <parameter description="Report on just registered info, or all info"
        name="registeredOnly" type="boolean"/>
    </signature>
  </operation>
```

```
<operation
  description="Perform diagnostics on the managed resource driven by current diagnostic mode setting."
  impact="ACTION" name="selfDiagnostic" role="operation"
  targetObjectType="objectReference" type="[Lcom.ibm.wsspi.rasdiag.DiagnosticEvent;"
      proxyInvokeType="multicall">
  <signature>
    <parameter description="Test ID to use"
      name="testId" type="java.lang.String"/>
    <parameter description="Report on just registered info, or all info"
      name="registeredOnly" type="boolean"/>
  </signature>
</operation>
<operation
  description="localize messages for console display"
  impact="INFO" name="localize" role="operation"
  targetObjectType="objectReference" type="[Ljava.lang.String;"
      proxyInvokeType="unicall">
  <signature>
    <parameter description="Message Keys" name="msgKeys" type="[Ljava.lang.String;"/>
    <parameter description="Locale to use for output" name="locale" type="java.util.Locale"/>
  </signature>
</operation>
```

Creating a Diagnostic Provider registration XML file

Programming knowledge of your system and the proper authorities to perform the following steps.

The steps that follow outline a general process for creating a Diagnostic Provider (DP) registration Extensible Markup Language (XML) file. The registration XML is used to provide information about the exposed configuration, state, and self diagnostic attributes and tests to the Diagnostic Provider utility. It is also used to populate objects needed later in the process, to assist in filtering, and to assist in localization.

1. Start with the DP document type definition (DTD). If you are using the helper methods (see the step called *Create your DP implementation* in “Creating a Diagnostic Provider” on page 137), you can use this DOCTYPE line to pick up the common DTD:

   ```
   <!DOCTYPE diagnosticProvider PUBLIC "RasDiag" "/DiagnosticProvider.dtd">
   ```

   If you are extending an existing MBean with an existing XML configuration, you might need either to add the DP XML to an existing DTD, or omit the DP XML entirely. If you omit the DP XML, you will not be able to validate that your XML file is well formed.
2. Follow the conventions described in “Conventions for Diagnostic Provider Extensible Markup Language” on page 138 to help keep your XML consistent with other components. You can find an example of a small DP registration XML file in “Diagnostic Provider method implementation” on page 140.

Associating a Diagnostic Provider ID with a logger

Components whose diagnostics are managed through a Diagnostic Provider MBean should include the Diagnostic Provider ID (DPID) in all logged messages. In some cases a single logger always logs with the same DPID. In those cases, it is appropriate to statically associate the DPID with the logger. In other cases, a logger might log on behalf of various diagnostic domains. For example, although every data source has a separate Diagnostic Provider MBean, they all share the same logger. In those cases, the DPID can be dynamically supplied on each logging call.

Static Assignment

The method below statically assigns a DPID to a logger.

Associate a DPID with a logger:

```
Logger logger = Logger.getLogger("com.ibm.ws.MyClass");
DiagnosticProviderHelper.addDiagnosticProviderIDtoLogger(logger, dpid);
```

Dynamic Assignment

DPIDs can be associated with a single log request by including them as the first message parameter, prefixed with **DPID:.** To associate a DPID with a single log request using a logger:

```
Object[] parms = new Object[] { "DPID:" + dpid };
logger.logp(classname, methodname, "MSG0001", parms);
```

Note that in the dynamic case, the DPID does not need to actually show up in the formatted message. The two examples below illustrate:

```
(in resource bundle)
// by not including {0} first parm is not printed in the message.
MSG0001=This message does not include the DPID.

// note - it is not recommended to print the DPID in your message.
MSG0002=This message includes the DPID...it's value is {0}.
```

It is recommended that messages not include the DPID in the formatted message. As shown above, this is done by not including {0} in the message value in the resource bundle.

Using Diagnostic Providers from wsadmin scripts

In addition to enabling Diagnostic Providers (DP) from the administration console, you can also use them through scripts from the Wsadmin tool. The procedure for doing so follows. For more detailed information about the Wsadmin tool see the scripting tool chapter in the *Administering applications and their environment* PDF book

1. List the MBeans that implement the Diagnostic Provider (DP) interface. Enter

   ```
   $AdminControl queryNames diagnosticProvider=true,*
   ```

   And you will see an output that displays all of the Diagnostic Providers in a format like this:

   ```
   "WebSphere:name=Default Datasource,process=server1,platform=dynamicproxy,node=
     camelhair,JDBCProvider=Derby JDBC Provider,
   diagnosticProvider=true,j2eeType=JDBCDataSource,J2EEServer=server1,Server=server1,
     version=6.1.0.0,type=DataSource,
   mbeanIdentifier=cells/camelhairCell/nodes/camelhair/servers/server1/resources.xml#
     DataSource_1131113688564,
   JDBCResource=Derby JDBC Provider,cell=camelhairCell"
   "WebSphere:name=DefaultEJBTimerDataSource,process=server1,platform=dynamicproxy,
     node=camelhair,
   JDBCProvider=Derby JDBC Provider (XA),diagnosticProvider=true,j2eeType=
     JDBCDataSource,J2EEServer=server1,Server=server1,version=6.1.0.0,type=DataSource,
     mbeanIdentifier=cells/camelhairCell/nodes/camelhair/servers/server1/
     resources.xml#DataSource_1000001,
   JDBCResource=Derby JDBC Provider (XA),cell=camelhairCell"
   WebSphere:name=WebcontainerDiagnosticProvider,process=server1,platform=
     dynamicproxy,node=camelhair,diagnosticProvider=true,
   version=6.1.0.0,type=WebcontainerEventProvider,mbeanIdentifier=null,
     cell=camelhairCell
   ```

2. Capture the ObjectName of your Diagnostic Provider in a variable. This enables you to reference your Diagnostic Provider more easily, especially in a script. For example, instead of typing all of those lines, if you want to work with the WebContainer Diagnostic Provider, for example, you can do the following:
   - set DP [lindex [$AdminControl queryNames name=WebcontainerDiagnosticProvider,diagnosticProvider=true,*] 0]

     This ObjectName stored in the DP variable can be used on the methods, or you can use the Diagnostic Provider name as text or a variable.
   - Now that you have the ObjectName in a variable, you can get the Diagnostic Provider name in a variable with the command:

     ```
     set DPNm [$AdminControl invoke $DS getDiagnosticProviderNameById $DP]
     ```

This provides the result:

```
WebContainerDP
```

Now the DiagnosticProvider (WebContainer) is addressable by its objectname in variable DP, or by its DiagnosticProvider name in variable DPNm. If you would prefer, you can hard-code the DPName *WebContainerDP* as it is short enough.

3. Save the ObjectName of the DiagnosticService MBean to a variable. For wsadmin, WebSphere Application Server provides this MBean so that the output of the Diagnostic Provider is more easily consumable. Enter

   ```
   set DS [lindex [$AdminControl queryNames name=DiagnosticService,*] 0]
   ```

4. Run a configDump. You can run a configDump and capture all attributes with the command:

   ```
   $AdminControl invoke $DS configDumpFormattedById [list $DP .* true null]
   ```

   This lists the values that the Diagnostic Provider used at start up (and possible current values). An excerpt of the configDump output follows.

Item Concatenated Name	Value
customProperties =	Null
defaultVirtualHostName =	default_host
jvmProps =	Null
localeProps =	Null
servletCachingEnabled =	false
aliases =	*:9080;*:80;*:9443;

5. Filter the output of your configDump. You can use configDumpFormatted (leaving off the ById) and switch **$DP** for **$DPNm** or the string **WebContainerDP**. This example uses *$DPNm* on this slightly modified version whereby it only picks up attributes dealing with automation:

   ```
   $AdminControl invoke $DS configDumpFormatted [list $DPNm .*auto.* true null]
   ```

   This results in just those attributes that contain **auto** in them. Full (but strict) regular expression syntax is allowed:

Item Concatenated Name	Value
autoLoadFiltersEnabled =	false
autoRequestEncoding =	false
autoResponseEncoding =	false
autoLoadFiltersEnabled =	false
autoRequestEncoding =	false
autoResponseEncoding =	false

The syntax is the same for stateDumps and selfDiagnostics

Viewing the run time configuration of a component using Diagnostic Providers

You must have sufficient authority to run the action.

You can use the panel associated with this task to navigate to the configuration data that can be used to check the health of a server runtime component. Runtime components that have associated diagnostic configuration data can include their Diagnostic Provider ID (DPID) in their log entries. If you know the DPID, you can enter it directly in the quick link text box. Otherwise, navigate to the desired process by using the tree view displayed at the bottom of the panel, as shown in the steps below.

1. Start the administration console.

2. From the task bar on the left side of the console, select **Troubleshooting**.
3. From the task bar on the left side of the console, select **Diagnostic Provider**.
4. From the task bar on the left side of the console, select **Configuration data**.
5. Either directly enter a Diagnostic Provider ID in the **Quick link using diagnostic provider ID** text box, or select a process (cluster / node / server) from the available processes displayed at the bottom of the panel under the section title **Server selection topology**.
6. From the list of available diagnostic providers for the selected process, choose the desired diagnostic provider name. The configuration data for that diagnostic provider appears.

Configuration data quick link or server selection

Use this panel to select a Diagnostic Provider server for viewing run time configuration data.

To view this administrative console page, click **Troubleshooting** > **Diagnostic Provider** > **Configuration data**

Quick link using Diagnostic Provider ID: From the Configuration data panel, enter a Diagnostic Provider ID to go directly to the page for the configuration data for the Diagnostic Provider for the specific server.

Server selection topology:

Use these folders to select server or cluster for viewing the configuration data for a Diagnostic Provider.

If you choose a cluster, whatever action you choose is performed on *each* server in the cluster.

The enterprise applications section shows you the servers that a particular application is running on. If you select a server from this list, the action is performed on that server, not specifically that application.

Diagnostic Providers (selection)

Use this panel to select a Diagnostic Provider from the selected server or cluster.

The list will contain only Diagnostic Providers registered on the selected server or cluster. Not all Diagnostic Providers register with every server in the cell.

You can follow several navigation paths to view this administrative console page. For example, click **Troubleshooting** > **Diagnostic Provider** > **Tests** > select a server or cluster name.

Name:

Choose a diagnostic provider from this list.

The path you chose to get to this panel determines which panel displays next.

- If you chose Troubleshooting > Diagnostic Provider > Tests, you see a panel that lists all of the available tests to run on the Diagnostic Provider.
- If you chose Troubleshooting > Diagnostic Provider > State data, you see a panel that shows the collected state data for the Diagnostic Provider.
- If you chose Troubleshooting > Diagnostic Provider > Configuration data, you see a panel that shows the configuration data for the Diagnostic Provider.

Configuration data

Use this panel to view the current configuration data for a Diagnostic Provider on a selected server or cluster. Not necessarily every piece of configuration data appears, but data that can be helpful in problem determination is shown.

You can follow several navigation paths to view this administrative console page. For example, click **Troubleshooting** > **Diagnostic Provider** > **Configuration data** > select a server or cluster name > select a Diagnostic Provider from the list.

The attributes show information that has been configured for the Diagnostic Provider. You can use the **Save** button to save the information to a file.

Note: Results from a configuration dump contain names that start with either *startup* or *current*. The *startup* entries represent data that was read in by the component at server startup time. The *current* entries contain data that is current – meaning the value of the attributes in use by the runtime at the time the configuration dump was requested.

Node:

This is the node name from where the configuration data was collected.

Server:

This is the server name from where the configuration data was collected.

Name:

This is the name of the attribute for the configuration data.

Value:

This is the value of the configuration data.

Description:

This is a description of the configuration data.

Viewing the run time state data or configuring the state data collection specifications for a Diagnostic Provider

You must have sufficient authority to execute the action.

You can use the panel associated with this task in two ways. You can navigate to the state data that can be used to check the health of a server runtime component, or you can configure the state data to be collected for a server. In the server selection topology section, use the view state data radio button to go to the list of registered diagnostic providers. Use the change state data collection specification radio button to modify the state collection specification for the runtime components for a server. Runtime components that have associated diagnostic state data can include their Diagnostic Provider ID (DPID) in their log entries. If you know the DPID, you can enter it directly in the quick link text box.

1. Start the administration console.
2. Select **Troubleshooting**.
3. Select **Diagnostic Provider**.
4. Select **State data**.
5. Select the **View state data** radio button to simply look at the state data, or select the **Change state data collection specification** radio button to change the configuration.
6. Either directly enter a Diagnostic Provider ID in the **Quick link using diagnostic provider ID** text box, or select a process (cluster / node / server) from the available processes displayed at the bottom of the panel.
   - If you chose the **View state data** radio button, a panel listing the available Diagnostic Providers appears. Choose one of the providers by clicking on it. A panel displaying the state data appears.

- If you chose the **Change state data collection specification** radio button, a panel appears that contains a list of the available Diagnostic Providers and a text entry block. The state collection specification for the selected process is managed from this panel. Select one of the available providers by using the checkbox next to it.

Diagnostic Provider State Collection Specification

In normal operation, most components should work optimally and not store any operational data that is not needed. There are times, however, when an administrator or automated tool may want a component to collect more information than normal to help in problem determination. This data could then be exposed through a State dump. The State Collection specification was created as a syntax for indicating what additional data the diagnostic providers in the system should retain.

For the syntax of the *aCollectionSpec* string, refer to the **DiagnosticConfigHome** API documentation. It is basically a semicolon (;) separated list of collection specification clauses which are of the form:

```
<DiagnosticProviderName regexp>:<AttributeId regexp>=[0|1]
```

Where the *DiagnosticProviderName* regular expression will make this clause apply to any Diagnostic Provider Name that matches that regular expression. The *AttributeId regexp* and the boolean value (**0** for off, and **1** for on) are stored in the *DiagnosticConfig* object that each Diagnostic Provider uses. Turning on or off, and processing the clauses left to right allows relatively complex specification. Any specification that is not explicitly turned **on** is considered to be off. This format is explained further in the following examples.

To turn on tracing for all attributes in the **MyDP** Diagnostic Provider:

```
MyDP:.*=1
```

To turn on tracing for *all* attributes of *all* Diagnostic Providers (this will probably impact system performance):

```
.*:.*=1
```

To turn on all tracing for all attributes of all Diagnostic Providers beginning with **ConnMgr** (for example, Data Sources):

```
ConnMgr.*:.*=1
```

This specification turns on special collection attributes in the **MyDP** Diagnostic Provider that begin with the string **PoolInfo**. If, however, the attribute begins with **PoolInfo.Db2Pool**, then the collection is off (because it is read left to right).

```
MyDP:PoolInfo.*=1;MyDP:PoolInfo.Db2Pool.*=0
```

It should be noted that State dumps can return important information even in the case where there is no State Collection Specification turned on for a Diagnostic Provider. Diagnostic Providers frequently have to keep some state information in order to operate. Anything in this category is available in a State dump even if there is no special data collection going on. Using the State Collection Specification may increase the amount of data available.

State Data Quick Link or Server Selection

Use this panel to select a server or cluster to either view collected state data, or to configure state data to collect for a Diagnostic Provider.

To view this administrative console page, click **Troubleshooting** > **Diagnostic Provider** > **State data**

Quick link using Diagnostic Provider ID:

Enter a Diagnostic Provider ID to go directly to the view page for the collected state data for the Diagnostic Provider.

Server selection topology:

Use these radio buttons and folders to select a specific server or cluster for viewing of state data or configuring the specification of state data.

If you choose a cluster, whatever action you choose is performed on *each* server in the cluster.

The enterprise applications section shows you the servers that a particular application is running on. If you select a server from this list, the action is performed on that server, not specifically that application.

View state data
Select this radio button to view the state data for a Diagnostic Provider. Then select the cell or cluster you want to work with.

Change state data collection settings
Select this radio button to configure the state collection specification for a Diagnostic Provider. Then select the cluster or managed server you want to work with.

State data

Use this panel to view the current state data for a Diagnostic Provider on a selected server or cluster.

To view this administrative console page, click **Troubleshooting** > **Diagnostic Provider** > **State data** > select the View state data radio button and then select a server or cluster name > select a Diagnostic Provider from the list.

The attributes show information that has been collected as part of the enabled state collection specification for the Diagnostic Provider. You can use the **save...** button to save the information to a file

Node:

This is the node name from where the state data was collected.

Server:

This is the server name from where the state data was collected.

Name:

This is the name of the state collection specification used to collect the state data.

Value:

This is the value of the state collection specification used to collect the state data.

Description:

This is a description of the state collection specification used to collect the state data.

Detailed state specification

Use this panel to view the attributes and descriptions of the Diagnostic Provider that you have selected.

To add attributes, select the checkbox next to your chosen diagnostic provider, then select the **Add to specification** button.

To remove a diagnostic provider's sub-component attribute from the state specification, select the sub-component attribute in the displayed list and then select the **Remove from specification** button.

When you are done adding or removing a diagnostic provider's sub-component attributes, select the **Done** button.

To view this administrative console page, click **Troubleshooting** > **Diagnostic Provider** > **State data** >select the View state data radio button and then select a server or cluster name > select a Diagnostic Provider from the list.

Attribute:

This is the individual state collection specification available for the Diagnostic Provider.

Description:

This is the description of the individual state collection specification item.

Change state specification

Use this panel to add a Diagnostic Provider and its attributes to the specification for collecting state data.

To add a diagnostic provider (DP) and *all* of its attributes, select the checkbox next to your chosen DP, then click on the **Add to specification** button. To add only *some* of the DP's attributes, click on the DP name itself in the list, and a new panel where you can perform this task appears.

To put the state specification into affect, select the **Apply** or **OK** button.

To reset the specification to its original state, use the **Reset** button.

To manually enter a state specification, update the text area with the state specification and use the **Update** button.

To view this administrative console page, click **Troubleshooting** > **Diagnostic Provider** > **State data** > select the Change state data collection specification radio button and then select a server or cluster name > select a Diagnostic Provider from the list.

Name:

This is a list of available Diagnostic Providers for the server selected.

Modifying the State Collection Specification from wsadmin scripts

In doing problem determination, you might want to begin collecting additional data during normal processing. This can be accomplished by modifying the State Collection Specification dynamically. This section illustrates how to do that through the Wsadmin tool. This technique can be used to turn on traces, as well as to turn off traces. Depending on the usage pattern of the component, the impact should take affect shortly after it is set. For more information on this tool, see the chapter, Wsadmin tool in the *Administering applications and their environment* PDF book.

1. Capture the DiagnosticService ObjectName into a variable. Enter

   ```
   set DS [lindex [$AdminControl queryNames name=DiagnosticService,*] 0]
   ```

2. Use this variable to drive the method to set the specification. Enter

   ```
   $AdminControl invoke $DS setStateCollectionSpec "SampleDiagnosticProvider:player.*=1;
   SampleDiagnosticProvider:defense.*=1"
   ```

   The specification is of the form **DiagnosticProviderName:AttributeId=0|1...** (with a semicolon at the end, multiple sub-specifications can be entered similar to the TraceSpec). The DiagnosticProviderName and AttributeId can be proper regular expressions.

Running a self diagnostic on a Diagnostic Provider

You must have sufficient authority to execute the action.

You can access a list of predefined diagnostic tests that you can use to check the status of a server runtime component. Runtime components that have associated diagnostic tests can include their Diagnostic Provide ID (DPID) in their log entries. If you know the DPID, you can enter it directly in the quick link text box. Otherwise, navigate to the desired process by using the tree view displayed at the bottom of the panel.

1. Start the administration console.
2. Select **Troubleshooting**.
3. Select **Diagnostic Provider**.
4. Select **Tests** .
5. Either directly enter a Diagnostic Provider ID in the **Quick link using diagnostic provider ID** text box, or select a process (cluster / node / server) from the available processes displayed in the **Server selection topology** section.
6. Select the desired self diagnostic test.
7. Read the output messages from the self diagnostic test.
8. Select a self diagnostic test message by clicking on it. The console displays a panel with the attributes related to the message you chose.

Tests Quick Link or Server Selection

Use this panel to select a Diagnostic Provider server for diagnostic tests.

To view this administrative console page, click **Troubleshooting** > **Diagnostic Provider** > **Tests**.

Quick link using Diagnostic Provider ID:

Enter a Diagnostic Provider ID to go directly to the view page for the collected state data for the Diagnostic Provider.

Server selection topology:

Use these folders to select server or cluster for viewing the available tests for a Diagnostic Provider.

If you choose a cluster, whatever action you choose is performed on *each* server in the cluster.

The enterprise applications section shows you the servers that a particular application is running on. If you select a server from this list, the action is performed on that server, not specifically that application.

Test selection

Use this panel to select one of the tests that are available for the chosen Diagnostic Provider on the chosen server or cluster.

You can follow several navigation paths to view this administrative console page. For example, click **Troubleshooting** > **Diagnostic Provider** > **Tests** > select a server or cluster name > select a Diagnostic Provider from the list.

Test identification:

Choosing a test ID causes the test to run. Results of the test are shown on the Test Results panel.

Test description:

A description of the test available to run on the Diagnostic Provider.

Test Results

Use this panel to see the results from the server or cluster members for the selected test.

You can follow several navigation paths to view this administrative console page. For example, click **Troubleshooting** > **Diagnostic Provider** > **Tests** > select a cluster name > select a Diagnostic Provider from the list > select a Test identification from the list.

Multiple results can be returned from a test from each server. The results are sorted by Node, then by Server, then by Severity. You can page through the messages that are returned.

Server:

The name of the server where the test result came back from.

Node:

The name of the node where the test result came back from.

Severity:

The severity of the result from the test run.

Message:

A description of the test result.

The entries in this column are linked to another panel. If you click on a message, you can see additional attributes associated with the message.

Test result details

Use this panel to see additional attributes for the selected test result.

To view this administrative console page, click **Troubleshooting** > **Diagnostic Provider** > **Tests** > select a cluster name > select a Diagnostic Provider from the list > select a Test identification from the list > select a message.

The attributes show information that helped to diagnose the condition described in the message. You can use the **Save** button to save to a file the attributes and the messages to which they correspond.

Name:

The name of the test.

Value:

This is the value of the test result.

Description:

This is a description of the test.

Chapter 7. Web applications

Web module or application server stops processing requests

Use this information to help determine why a Web module or application server has stopped processing new requests.

If an application server's process spontaneously closes, or its Web modules stop responding to new requests:

- Isolate the problem by installing Web modules on different servers, if possible.
- You can use the Tivoli performance viewer to determine which resources have reached their maximum capacity, such as Java heap memory (indicating a possible memory leak) and database connections. If a particular resource appears to have reached its maximum capacity, review the application code for a possible cause:
  - If database connections are used and never freed, ensure that application code performs a **close()** on any opened **Connection** object within a **finally{}** block.
  - If there is a steady increase in servlet engine threads in use, review application **synchronized** code blocks for possible deadlock conditions.
  - If there is a steady increase in a JVM heap size, review application code for memory leak opportunities, such as static (class-level) collections, that can cause objects to never get garbage-collected.

  See the *Tuning guide* PDF for more information about this tool.
- As an alternative to using the performance viewer to detect memory leak problems, enable verbose garbage collection on the application server. This feature adds detailed statements to the JVM error log file of the application server about the amount of available and in-use memory. To set up verbose garbage collection:
  1. Select **Servers > Application Servers >** *server_name* **>Java and Process Management > Process Definition > Java Virtual Machine**, and enable **Verbose Garbage Collection**.
  2. Stop and restart the application server.
  3. Periodically, or after the application server stops, browse the log file for garbage collection statements. Look for statements beginning with ″allocation failure″. The string indicates that a need for memory allocation has triggered a JVM garbage collection (freeing of unused memory). Allocation failures themselves are normal and not necessarily indicative of a problem. The allocation failure statement is followed by statements showing how many bytes are needed and how many are allocated.

     If there is a steady increase in the total amount of free and used memory (the JVM keeps allocating more memory for itself), or if the JVM becomes unable to allocate as much memory as it needs (indicated by the bytes needed statement), there might be a memory leak.
- Force an application to create a thread dump (or javacore). Here is the process for forcing a thread dump, which is different from the process in earlier releases of the product:
  1. Using the wsadmin command prompt, get a handle to the problem application server:

     ```
     wsadmin>set jvm [$AdminControl completeObjectName type=JVM,process=server1,*]
     ```
  2. Generate the thread dump:

     ```
     wsadmin>$AdminControl invoke $jvm dumpThreads
     ```
  3. Look for an output file in the *profile_root*/logs directory with a name like javacore.*jobnum.jobuser.jobname.timestamp*.txt.
- Browse the thread dump for clues:
  - The thread dump shows information on the current Java heap size, and the minimum and maximum heap size settings. Look for an excessive current heap size.
  - The thread dump contains a snapshot of each thread in the process, starting in the section labeled ″Thread Information.″
    - Look for threads that are waiting on locks held by other threads.

- Look for multiple threads in the same Java application code source location. Multiple threads from the same location might indicate a deadlock condition (multiple threads waiting on a monitor) or an infinite loop, and help identify the application code with the problem.

  It is normal for certain components in the WebSphere Application Server runtime to have certain types of threads in the same Java code source location. These components include the Web container, the EJB container and the ORB thread pool.

IBM Support has documents and tools that can save you time gathering information needed to resolve problems as described in “Troubleshooting help from IBM” on page 126. Before opening a problem report, see the Support page:

- http://www.ibm.com/servers/eserver/support/iseries/software/v5r3/index.html

Errors starting an application

Use this information for troubleshooting problems that occur when starting an application.

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

HTTP server and Application Server are working separately, but requests are not passing from HTTP server to Application Server

If your HTTP server appears to be functioning correctly, and the Application Server also works on its own, but browser requests sent to the HTTP server for pages are not being served, a problem exists in the WebSphere Application Server plug-in.

In this case:

1. Determine whether the HTTP server is attempting to serve the requested resource itself, rather than forwarding it to the WebSphere Application Server.
   a. Browse the HTTP server access log (*IHS install root*/logs/access.log for IBM HTTP Server). It might indicate that it could not find the file in its own document root directory.
   b. Browse the plug-in log file as described below.
2. Browse the *plugin_install_root*/logs/*web_server_name*/http_plugin.log file for clues to the problem. Make sure the timestamps with the most recent plug-in information stanza, which is printed out when the plug-in is loaded, correspond to the time the Web server started.
3. Turn on plug-in tracing by setting the LogLevel attribute in the plugin-cfg.xml file to Trace and reloading the request. Browse the *plugin_install_root*/logs/*Web_server_name*/http_plugin.log file. You should be able to see the plug-in attempting to match the request URI with the various URI definitions for the routes in the plugin-cfg.xml. Check which rules the plug-in is not matching against

and then figure out if you need to add additional ones. If you just recently installed the application you might need to manually regenerate the plug-in configuration to pick up the new URIs related to the new application.

For further details on troubleshooting plug-in-related problems, see Webserver plug-in troubleshooting tips located in the *Administering applications and their environment* PDF book.

File serving problems

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

- Verify that your files are in the right place: the **document root** directory of your Web application WebSphere Application Server follows the J2EE standard, which means that the document root is the *Web_module_name*.war directory of your deployed Web application.

  Typically this directory will be found in the *profile_root*/installedApps/*nodename*/*appname*.ear directory or *profile_root*/installedApps/*nodename*/*appname*Network.ear directory.

  If the files are in a subdirectory of the document root, verify that the reference to the file reflects that. That is, if the invoices.html file is stored in Windows directory *Web_module_name*.war\invoices, then links from other pages in the Web application to display it should read `"invoices\invoices.html"`, not `"invoices.html"`.
- Verify that your Web application is configured to enable file serving (in other words, that it is enabled to display static resources like image and `.html` files):
  1. View the file serving property of the hosting Web module by browsing the source `.war` file in an assembly tool. If necessary, update the property and redeploy the module. For more information about the assembly tool, refer to the assembly tools section of the *Developing and deploying applications* PDF book.
  2. Edit the **`fileServingEnabled`** property in the deployed Web application `ibm-web-ext.xmi` configuration file.

     The file typically is found in the *`profile_root`*`/config/cells/`*`nodename`* or *`nodename`*`Network/applications/`*`application_name`*`/deployments/`*`application name/Web_module_name`*`/web-inf` directory.

Graphics do not appear in the JSP file or servlet output

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

- Verify that your graphic files are in the right place: the **document root** directory of your Web application. WebSphere Application Server Version 5 follows the J2EE standard, which means that the document root is the *Web_module_name*.war directory of your deployed Web application.

  Typically, this directory is found in the *profile_root*/installedApps/*nodename*/*appname*.ear directory or *profile_root*/installedApps/*nodename*/*appname*Network.ear directory.

  If the graphics files are in a subdirectory of the document root, verify that the reference to the graphic reflects that; for example, if the `banner.gif` file is stored in Windows directory *Web_module_name*.war/images, the tag to display it should read: **`<img SRC="images/banner.gif">`**, not `<img SRC="banner.gif">`.
- Verify that your Web application is configured to enable file serving (that is, display of static resources like image and `.html` files).
  1. View the file serving property of the hosting Web module by browsing the source `.war` file in an assembly tool. If necessary, update the property and redeploy the module. For more information about the assembly tool, refer to the assembly tools section of the *Developing and deploying applications* PDF book.
  2. Edit the **`fileServingEnabled`** property in the deployed Web application `ibm-web-ext.xmi` configuration file.

     The file typically is found in the *`profile_root`*`/config/cells/`*`nodename`* or *`nodename`*`Network/applications/`*`application_name`*`/deployments/`*`application name/Web_module_name`*`/web-inf` directory.

3. After completing the previous step:
   – In the administrative console, expand the **Environment** tree control .
   – Click **Update WebSphere Plugin**.
   – Stop and restart the HTTP server and retry the Web request.

SRVE0026E: [Servlet Error]-[Unable to compile class for JSP file

If this error appears in a browser when trying to access a new or modified .jsp file for the first time, the most likely cause is that the JSP file Java source failed (was incorrect) during the javac compilation phase.

Check the SystemErr.log file for a compiler error message, such as:

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

Fix the problem in the JSP source file, save the source and request the JSP file again.

If this error occurs when trying to serve a JSP file that was copied from another system where it ran successfully, then there is something different about the new server environment that prevents the JSP file from running. Browse the text of the error for a statement like:

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

This error indicates that a supporting class or jar file is not copied to the target server, or is not on the class path. Find the MyClass.class file, and place it on the Web module WEB-INF/classes directory, or place its containing .jar file in the Web module WEB-INF/lib directory.

Verify that the URL used to access the resource is correct by doing the following:

- For a JSP file, `html` file, or image file: **`http://host_name/Web_module_context_root/subdir under doc root, if any/filename.ext`**. The document root for a Web application is the *application_name*.WAR directory of the installed application.
  – For example, to access the myJsp.jsp file, located in c:\WebSphere\ApplicationServer\installedApps\myEntApp.ear\myWebApp.war\invoices on myhost.mydomain.com, and assuming the context root for the myWebApp Web module is myApp, the URL is `http://myhost.mydomain.com/myApp/invoices/myJsp.jsp`.
  – JSP serving is enabled by default. File serving for HTML and image files must be enabled as a property of the Web module, in an assembly tool, or by setting the **`fileServingEnabled`** property to **true** in the `ibm-web-ext.xmi` file of the installed Web application and restarting the application. For more information about the assembly tool, refer to the assembly tools section of the *Developing and deploying applications* PDF book.
- For servlets served by class name, the URL is `http://`*hostname*`/`*Web_module_context_root*`/servlet/`*packageName.className*.

  For example, to access myCom.myServlet.class, located in profile_root/installedApps/myEntApp.ear/myWebApp.war/WEB-INF/classes, and assuming the context root for the myWebApp module is ″myApp″, the URL would be `http://myhost.mydomain.com/myApp/servlet/myCom.MyServlet`.
- Serving servlets by class name must be enabled as a property of the Web module, and is enabled by default. File serving for HTML and image files must be enabled as a property of the Web application, in an assembly tool, or by setting the **`fileServingEnabled`** property to **true** in the **`ibm-web-ext.xmi`** file of the installed Web application and restarting the application. For more information about the assembly tool, refer to the assembly tools section of the *Developing and deploying applications* PDF book.

Correct the URL in the ″from″ HTML file, servlet or JSP file. An HREF with no leading slash (/) inherits the calling resource context. For example:

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

After modifying and saving a JSP file, the change does not show up in the browser (the old JSP file displays)

It is probable that the Web application is not configured for servlet reloading, or the reload interval is too high.

To correct this problem, in an assembly tool, check the **Reloading Enabled** flag and the **Reload Interval** value in the IBM Extensions for the Web module in question. Enable reloading, or if it is already enabled, then set the Reload Interval lower. For more information about the assembly tool, refer to the assembly tools section of the *Developing and deploying applications* PDF book.

Message like "Message: /jspname.jsp(9,0) Include: Mandatory attribute page missing" appears when attempting to browse JSP file

It is probable that the JSP file failed during the translation to Java phase. Specifically, a JSP directive, in this case an Include statement, was incorrect or referred to a file that could not be found.

To correct this problem, fix the problem in the JSP source, save the source and request the JSP file again.

The Java source generated from a JSP file is not retained in the temp directory (only the class file is found)

It is probable that the JSP processor is not configured to keep generated Java source.

In an assembly tool, check the **JSP Attributes** under **Assembly Property Extensions** for the Web module in question. Make sure the **keepgenerated** attribute is there and is set to `true`. If not, set this attribute and restart the Web application. To see the results of this operation, delete the class file from the temp directory to force the JSP processor to translate the JSP source into Java source again. For more information about the assembly tool, refer to the assembly tools section of the *Developing and deploying applications* PDF book.

The JSP Batch Compiler fails with the message "Enterprise Application [application name you typed in] not found."

It is probable that the full enterprise application path and name, starting with the `.ear` subdirectory that resides in the `applications` directory is expected as an argument to the JspBatchCompiler tool, not just the display name.

The directory path is *`profile_root`*`/config/cells/`*`node_name`*`Network/applications`.

For example:
- `"JspBatchCompiler -enterpriseapp.name sampleApp.ear/deployments/sampleApp"` is correct, as opposed to
- `"JspBatchCompiler -enterpriseapp.name sampleApp"`, which is incorrect.

There is a translation problem with non-English browser input

If non-English-character-set browser input cannot be translated after being read by a servlet or JSP file, ensure that the request parameters are encoded according to the expected character set before reading. For example, if the site is Chinese, the target `.jsp` file should have a line:

```
req.setCharacterEncoding("gb2312");
```

before any req.getParameter method calls.

This problem affects servlets and `jsp` files ported from earlier versions of WebSphere Application Server, which converted characters automatically based upon the locale of the WebSphere Application Server.

Scroll bars do not appear around items in the browser window

In some browsers, tree or list type items that extend beyond their allotted windows do not have scroll bars to permit viewing of the entire list.

To correct this problem, right-click on the browser window and click **Reload** from the menu.

Error ″Page cannot be displayed... server not found or DNS error″ appears when attempting to browse a JavaServer Pages (JSP) file using Internet Explorer

This error can occur when an HTTP timeout causes the servant to be brought down and restarted. To correct this problem, increase the ConnectionIOTimeOut value:

1. From the administrative console, select **System administration > Deployment manager > Administration Services > Custom Properties**
2. Select ConnectionIOTimeOut.
3. Increase the ConnectionIOTimeOut value.
4. Click **OK**.

A Web resource does not display

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

1. Verify that your HTTP server is healthy by accessing the URL `http://`*`server_name`* from a browser and seeing whether the Welcome page appears. This action indicates whether the HTTP server is up and running, regardless of the state of WebSphere Application Server.
2. If the HTTP server Welcome page does not appear, that is, if you get a browser message like `page cannot be displayed` or something similar, try to diagnose your Web server problem.
3. If the HTTP server appears to function, the Application Server might not be serving the target resource. Try accessing the resource directly through the Application Server instead of through the HTTP server.

   If you cannot access the resource directly through the Application Server, Verify that the URL used to access the resource is correct.

   If the URL is incorrect and it is created as a link from another JSP file, servlet, or HTML file, try correcting it in the browser URL field and reloading, to confirm that the problem is a malformed URL. Correct the URL in the ″from″ HTML file, servlet or JSP file.

   If the URL appears to be correct, but you cannot access the resource directly through the Application Server, verify the health of the hosting Application Server and Web module:
   a. View the hosting Application Server and Web module in the administrative console to verify that they are up and running.
   b. Copy a simple HTML or JSP file (such as `SimpleJsp.jsp` in the WebSphere Application Server directory structure) to your Web module document root, and try to access it. If successful, the problem is with your resource.

View the JVM log of your Application Server to find out why your resource cannot be found or served .

4. If you can access the resource directly through the Application Server, but not through an HTTP server, the problem lies with the HTTP plug-in -- the component that communicates between the HTTP server and the WebSphere Application Server.
5. If the JSP file and the servlet output are served, but not static resources such as `.html` and image files, see the steps for enabling file serving.
6. If some kinds of resources display correctly, but you cannot display a servlet by its class name:
   - Verify that the servlet is in a directory in the Web module class path, such as in the /*Web_module_name*.war/WEB-INF/classes directory.
   - Verify that you specify the full class name of the servlet, including its package name, in the URL.
   - Verify that `"/servlet"` precedes the class name in the URL. For example, if the root context of a Web module is `"myapp"`, and the servlet is `com.mycom.welcomeServlet`, then the URL reads:

     `http://`*`hostname`*`/myapp/servlet/com.mycom.welcomeServlet`
   - Verify that serving the servlets by class name is enabled for the hosting Web module by opening the source Web module in an assembly tool and browsing the *serve servlets by classname* setting in the IBM Extensions property page. If necessary, enable this flag and redeploy the Web module. For more information about the assembly tool, refer to the assembly tools section of the *Developing and deploying applications* PDF book.
   - For servlets or other resources served by mapped URLs, the URL is http://*hostname*/*Web module context root*/*mappedURL*.

If none of these steps fixes your problem, see if the problem has been identified and documented by looking at available online support (hints and tips, technotes, and fixes). If you do not find your problem listed there, see “Troubleshooting help from IBM” on page 126.

Diagnosing Web server problems

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

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

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

If the HTTP server does not start:

- Examine the HTTP server error log for clues.
- Try restoring the HTTP server to its configuration prior to installing WebSphere Application Server and restarting it. If you are using IBM HTTP Server:
  - Rename the file *IHS_install_dir*\httpd.conf.
  - Copy the httpd.conf.default file to the httpd.conf directory.
  - If Apache is running, stop and restart it.
- For the Sun ONE (iPlanet) Web server, restore the obj.conf configuration file for Sun ONE V4.1 and both obj.conf and magnus.conf files for Sun ONE V6.0 and later.
- For the Microsoft Internet Information Server (IIS), remove the WebSphere Application Server plug-in through the IIS administrative GUI.

If restoring the HTTP server default configuration file works, manually review the configuration file that has WebSphere Application Server updates to verify directory and file names for WebSphere Application Server files. If you cannot manually correct the configuration, you can uninstall and reinstall WebSphere Application Server to create a clean HTTP configuration file.

If restoring the default configuration file does not help, contact technical support for the Web server you are using. If you are using IBM HTTP Server with WebSphere Application Server, check available online support (hints and tips, technotes, and fixes). If you do not find your problem listed there, see "Troubleshooting help from IBM" on page 126

Accessing a Web resource through the application server and bypassing the HTTP server

Starting with WebSphere Application Server Version 4.0, you can bypass the HTTP server and access a Web resource through the application server. It is not recommended to serve a production Web site in this way, but it provides a good diagnostic tool when it is not clear whether a problem resides in the HTTP server, WebSphere Application Server, or the HTTP plug-in.

To access a Web resource through the Application Server:
1. Determine the port of the HTTP service in the target application server.
   a. In the WebSphere administrative console, click **Servers>Manage Application Servers**.
   b. Select the target server, then under Additional Properties click **Web Container**.
   c. Under the Additional Properties of the Web container, click **HTTP Transports**. You see the ports listed for virtual hosts served by the application server.
   d. There can be more than one port listed. In the default application server (server1), for example, 9060 is the port reserved for administrative requests, 9443 and 9043 are used for SSL-encrypted requests. To test the sample `"snoop"` servlet, for example, use the default application port 9080, unless it changes.
2. Use the HTTP transport port number of the application server to access the resource from a browser. For example, if the port is 9080, the URL is `http://hostname:9080/myAppContext/myJSP.jsp`.
3. If you are still unable to access the resource, verify that the HTTP transport port is in the "Host Alias" list:
   a. Click **Application Servers > Your_ApplicationServer > Web Container > HTTP Transports** to check the Default virtual host and the HTTP transport ports used by this application server.
   b. Click **Environment > Manage Virtual Hosts > default host > Host Aliases** to check if the HTTP transport port exists. Add an entry if necessary. For example, if the HTTP port for your application is server is 9080, add a host alias of *:9082.

JavaServer Pages troubleshooting tips

Use this tips to troubleshoot problems with JavaServer Pages.

JavaServer Pages source code shown by the Web server

If you share the document root of the WebSphere Application Server with the Web server document root, a security exposure can result as the Web server might display the JavaServer Pages (JSP) source file as plain text.

Problem

You can use the WebSphere Web server plug-in set of rules to determine whether a given request will be handled by the WebSphere Application Server. When an incoming request fails to match those rules, the Web server plug-in returns control to the Web server so that the Web server can fulfill the request. In this case, the unknown host header causes the Web server plug-in to return control to the Web server because the rules do not indicate that the WebSphere Application Server should handle it. Therefore, the Web server looks for the request in the Web server document root. Since the JSP source file is stored in the document root of the Web server, the Web server finds the file and displays it as plain text.

Suggested solution

Move the WebSphere Application Server JSP source file outside of the Web server document root. Then, when this request comes in with the unknown host header, the plug-in returns control to the Web server and the JSP source file is not found in the document root. Therefore, the Web server returns a `404 File Not Found` error rather than the JSP source file.

Problems displaying double-byte character set (DBCS) characters when using the @include directive

JavaServer Pages files that use the @include directive might experience problems when displaying double-byte character set (DBCS) characters. Some applications that are migrated to WebSphere Application Server Version 6.0 and above might need to be modified to comply with the JSP 2.0 specification as a result of backwards compatibility issues. The JSP 2.0 specification requires that each statically included resource must set a page encoding or content type because the character encoding for each file is determined separately, even if one file includes another using the include directive.

Problems using the JavaServer Pages (JSP) engine

If you are having difficulty using the JavaServer Pages (JSP) engine, try these steps:

1. Determine whether other resources such as `.html` files or servlets are being requested and displayed correctly. If they are not, the problem probably lies at a deeper level, such as with the HTTP server.
2. If other resources are being displayed correctly, determine whether the JSP processor has started normally:
   - Browse the JVM logs of the server hosting the JSP files you are trying to access. The following messages indicate that the JSP processor has started normally:

     ```
     Extension Processor [class com.ibm.ws.jsp.webcontainerext.JSPExtensionProcessor]
     was initialized successfully.
     Extension Processor [class com.ibm.ws.jsp.webcontainerext.JSPExtensionProcessor]
     has been associated with patterns [*.jsp *.jspx *.jsw *.jsv ].
     ```

     If the JSP processor fails to load, you will see a message such as

     ```
     No Extension Processor found for handling JSPs.
     JSP Processor not defined. Skipping : jspfilename.
     ```

     in the *root_dir*/logs/*server_name*/SystemOut.log file
3. If the JSP engine has started normally, the problem may be with the JSP file itself.
   - The JSP may have invalid JSP syntax and could not be processed by the JSP Processor. Examine the *root_dir*/logs/*server_name*/SystemOut.log file of the target application for invalid JSP directive syntax messages. Errors similar to the following in a browser indicate this kind of problem:

     ```
     Message: /filename.jsp(2,1)JSPG0076E: Missing required attribute page for jsp
                 element jsp:include
     ```

     This example indicates that line 2, column 1 of the named JavaServer Pages file is missing a mandatory attribute for the jsp:include action. Similar messages are displayed for other syntax errors.
   - Examine the target application server's `SystemErr.log` files for problems with invalid Java syntax. Errors similar to **`Message: Unable to compile class for JSP`** in a browser indicate this kind of problem.

     The error message output from the Javac compiler will be found in the `SystemErr.log`. It might look like:

     ```
     JSPG0091E: An error occurred at line: 2 in the file: /myJsp.jsp
     JSPG0093E: Generated servlet error: c:\WASROOT\temp\ ...
     test.war\_myJsp.java:16: myInt is already defined in com.ibm.ws.jsp20._myJsp
     int myInt = 122; String myString = "number is 122"; static int myStaticInt=22;
     int myInt=121;
              ^  1 error
     ```

     Correct the error in the JSP file and retry the file.

- Examine the target application server's server log files files for problems with invalid Java syntax. Errors similar to **Message: Unable to compile class for JSP** in a browser indicate this kind of problem.

  The error message output from the Javac compiler will be found in the SystemErr.log. It might look like:

```
JSPG0091E: An error occurred at line: 2 in the file: /myJsp.jsp
JSPG0093E: Generated servlet error: c:\WASROOT\temp\ ...
test.war\_myJsp.java:16: myInt is already defined in com.ibm.ws.jsp20._myJsp
int myInt = 122; String myString = "number is 122"; static int myStaticInt=22;
int myInt=121;
         ^  1 error
```

  Correct the error in the JSP file and retry the file.

JavaServer Pages fail to compile when using precompile

Symptom

JavaServer Pages fail to compile during deployment through the administrative console when precompile is selected.

```
SystemErr R com.ibm.websphere.management.exception.AdminException:
ADMA0021E: Error in compiling jsps - xyz.war (rc=1)
```

Problem

JavaServer Pages fail to compile during deployment through the administrative console when precompile is selected when there is a dependency on another Java archive (JAR) file that is not available on any class path.

Suggested solution

You may use wsadmin scripting to precompile JSP files during enterprise application deployment. However if you want to use the administrative console, then compile all JSP files before packaging the application.

1. Add the dependent JAR to the deployment manager in a cell environment.
   a. Click **System Administration > Deployment manager > Java and Process Management > Process Definition > Java Virtual Machine** in the console navigation.
   b. Add fully qualified dependent JAR in class path field.
   c. Click OK.
   d. Restart deployment manager.
2. Add the dependent JAR to the application server.
   a. Click **Servers > Application servers > *server1* > Java and Process Management > Process Definition > Java Virtual Machine** in the console navigation.
   b. Add fully qualified dependent JAR in class path field.
   c. Click OK.
   d. Restart application server.

JSPG0089E: Mismatch found between page directive encoding Shift_JIS and xml prolog encoding UTF-8

Symptom

The following error appears:

```
JSP Processing Error

HTTP Error Code: 500

Error Message: /test.jsp(2,1) /test.jsp(2,1) JSPG0089E: Mismatch found between
page directive encoding Shift_JIS and xml prolog encoding UTF-8
```

Problem

The pageEncoding attribute in the jsp:directive.page element is not UTF-8.

Suggested solution

JavaServer Pages must specify a prolog that matches the encoding specified in the page directive. For example,

```
<?xml version="1.0" encoding="Shift_JIS"?>
<jsp:root xmlns:jsp="http://java.sun.com/JSP/Page" version="2.0">
<jsp:directive.page language="java" contentType="text/html";
 charset=Shift_JIS" pageEncoding="Shift_JIS"/>
<jsp:text>XXXXXjsp:text>XXXXX>
</jsp:root>
```

For additional information, see section JSP.4.1, Page Character Encoding, in the JavaServer Pages specification and section 4.3.3 and appendix F.1 of the Extensible Markup Language (XML) specification

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

For current information available from IBM Support on known problems and their resolution, see the IBM Support page. The IBM Support page contains documents that can save you time gathering information needed to resolve this problem.

Web container troubleshooting tips

If you are having problems starting a Web module, or accessing resources within a particular Web module:

- View the JVM logs and process logs for the application server which hosts the problem Web modules, and look for messages in the JVM output file which indicate that the web module has started successfully. You should see messages similar to the following:

  ```
  WebContainer  A SRVE0161I: IBM WebSphere Application Server - Web Container.
  Copyright IBM Corp. 1998-2002
  WebContainer  A SRVE0169I: Loading Web Module: [module_name]
  ApplicationMg A WSVR0221I: Application started: [application_name]
  HttpTransport A SRVE0171I: Transport http is listening on port [port_number]
  [server_name] open for e-business in profile_root/logs/[server_name]/SystemOut.log
  ```
- For specific problems that can cause servlets, HTML files, and JavaServer Pages (JSP) files not to be served, see Web resource (JSP file, servlet, HTML file, image) does not display .
- For a detailed trace of the run-time behavior of the Web container, enable trace for the component com.ibm.ws.webcontainer using `com.ibm.ws.webcontainer.*=all:com.ibm.ws.wswebcontainer.*=all`.

If application server related calls fail during Servlet.init method, you can either:

- Initialize the servlet manually by making a single request to that servlet in your browser when the server is ready for e-business instead of starting the servlet upon startup or
- You can choose not to make application server related calls in the servlet's init method.

If the property to start servlets during application server startup is enabled, part of its startup process calls the Servlet.init method on its servlets when you start the Web container. Therefore, when the Web container is starts and calls the init method, other components such as Naming and Work Load Management may not be fully started yet. As a result, application server related calls may not work since all of the application server components may not be ready yet. Once the application server is 'ready for e-business', it is completely ready.

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

For current information available from IBM Support on known problems and their resolution, see the IBM Support page.

IBM Support has documents that can save you time gathering information needed to resolve this problem. Before opening a PMR, see the IBM Support page.

Troubleshooting tips for Web application deployment

Deployment of a Web application is successful if you can access the application by typing a Uniform Resource Locator (URL) in a browser, or if you can access the application by following a link.

If you cannot access your application, follow these steps to eliminate some common errors that can occur during migration or deployment.

Web module does not run in WebSphere Application Server Version 5.x or 6.x

Symptom Your Web module does not run when you migrate it to Version 5.x or 6.x

Problem In Version 4.x, the classpath setting that affected visibility was *Module Visibility Mode*. In Versions 5.x and 6.x, you must use class loader policies to set visibility.

Recommended response Reassemble an existing module, or change the visibility settings in the class loader policies.

See Class loaders and Class loading for more information.

Welcome page is not visible.

Symptom You cannot access an application with a Web path of:

```
/webapp/myapp
```

Problem The default welcome page for a Web application is assumed to be *index.html*. You cannot access the default page of the *myapp* application unless it is named *index.html*.

Recommended response To identify a different welcome page, modify the properties of the Web module during assembly, see the topic, Assembling Web applications for more information in the *Developing and deploying applications* PDF book.

HTML files are not found.

Symptom Your Web application ran successfully on prior versions, but now you encounter errors that the welcome page (typically *index.html*), or referenced HTML files are not found:

```
Error 404: File not found: Banner.html
Error 404: File not found: HomeContent.html
```

Problem

For security and consistency reasons, Web application URLs are now case-sensitive on all operating systems.

Suppose the content of the index page is as follows:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 5.0 Frameset//EN">
<HTML>
<TITLE>
Insurance Home Page
</TITLE>
    <frameset   rows="18,80">
    <frame      src="Banner.html"           name="BannerFrame"  SCROLLING=NO>
    <frame      src="HomeContent.html"      name="HomeContentFrame">
    </frameset>
</HTML>
```

However the actual file names in the `\WebSphere\AppServer\installedApps\...` directory where the application is deployed are:

```
banner.html
homecontent.html
```

Recommended response

To correct this problem, modify the *index.html* file to change the names *Banner.html* and *HomeContent.html* to *banner.html* and *homecontent.html* to match the names of the files in the deployed application.

For current information available from IBM Support on known problems and their resolution, see the IBM Support page.

IBM Support has documents that can save you time gathering information needed to resolve this problem. Before opening a PMR, see the IBM Support page.

HTTP session manager troubleshooting tips

This article provides troubleshooting tips for problems creating or using HTTP sessions with your Web application hosted by WebSphere Application Server.

Here are some steps to take:

- See HTTP session aren't getting created or are getting dropped to see if your specific problem is discussed.
- View the JVM logs for the application server which hosts the problem application:
  - first, look at messages written while each application is starting. They will be written between the following two messages:

    ```
    Starting application: application
    ..................
    Application started: application
    ```
  - Within this block, look for any errors or exceptions containing a package name of com.ibm.ws.webcontainer.httpsession. If none are found, this is an indication that the session manager started successfully.
  - Error ″**SRVE0054E: An error occurred while loading session context and Web application**″ indicates that SessionManager didn't start properly for a given application.
  - Look within the logs for any Session Manager related messages. These messages will be in the format SESNxxxxE and SESNxxxxW for errors and warnings, respectively, where xxxx is a number identifying the precise error. Look up the extended error definitions in the Session Manager message table.
- See the Best practices for using HTTP Sessions section in the *Developing and deploying applications* PDF books for more details.
- To dynamically view the number of sessions as a Web application is running, enable performance monitoring for HTTP sessions. This will give you an indication as to whether sessions are actually being created.

- To learn how to view the HTTP session counters as the application runs, read the Monitoring performance with Tivoli Performance Viewer chapter of the *Administering applications and their environment* PDF book.
- Alternatively, a special servlet can be invoked that displays the current configuration and statistics related to session tracking. This servlet has all the counters that are in performance monitor tool and has some additional counters.
  - Servlet name: **com.ibm.ws.webcontainer.httpsession.IBMTrackerDebug**.
  - It can be invoked from any Web module which is enabled to serve by class name. For example, using default_app, **http://localhost:9080/servlet/ com.ibm.ws.webcontainer.httpsession.IBMTrackerDebug**.
  - If you are viewing the module via the serve-by-class-name feature, be aware that it may be viewable by anyone who can view the application. You may wish to map a specific, secured URL to the servlet instead and disable the serve-servlets-by-classname feature.
- Enable tracing for the HTTP Session Manager component:
  - Use the trace specification **com.ibm.ws.webcontainer.httpsession.*=all=enabled**. Follow the instructions for dumping and browsing the trace output to narrow the origin of the problem.
  - If you are using persistent sessions based on memory replication, also enable trace for **com.ibm.ws.drs.***.
- If you are using **database-based persistent sessions**, look for problems related to the **data source** the Session Manager relies on to keep session state information. For details on diagnosing database related problems see the Errors accessing a datasource or connection pool in the *Administering applications and their environment* PDF book

Error message `SRVE0079E Servlet host not found` after you define a port

Error message SRVE0079E can occur after you define the port in WebContainer > HTTP Transports for a server, indicating that you do not have the port defined in your virtual host definitions. To define the port,

1. On the administrative console, go to Environment > Virtual Hosts > default_host> Host Aliases> New
2. Define the new port on host ″*″

The application server gets EC3 - 04130007 ABENDs

To prevent an EC3 - 04130007 abend from occuring on the application server, change the HTTP Output timeout value. The custom property *ConnectionResponseTimeout* specifies the maximum number of seconds the HTTP port for an individual server can wait when trying to read or write data. For instructions on how to set *ConnectionResponseTimeout*, see HTTP transport custom properties section of the *Administering applications and their environment* PDF book.

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

For current information available from IBM Support on known problems and their resolution, see the IBM Support page.

IBM Support has documents that can save you time gathering information needed to resolve this problem. Before opening a PMR, see the IBM Support page.

Problems creating or using HTTP sessions

This article provides troubleshooting information related to creating or using Hypertext Transfer Protocol (HTTP) sessions.

To view and update the session manager settings discussed here, use the administrative console. Select the application server that hosts the problem application, then under **Additional properties**, select **Web Container**, then **Session manager**.

What kind of problem are you having?
- HTTP Sessions are not getting created, or are lost between requests.
- HTTP Sessions are not persistent (session data lost when application server restarts, or not shared across cluster).
- Session is shared across multiple browsers on same client machine.
- Session is not getting invalidated immediately after specified session timeout interval.
- Unwanted sessions are being created by JavaServer Pages.
- Session data intended for one client is seen by another client.
- A ClassCastException error occurs during failover of a session that contains an Enterprise JavaBeans (EJB) reference.

If your problem is not described here, or none of these steps fixes the problem:
- Review "HTTP session manager troubleshooting tips" on page 167 for general steps on debugging session-manager related problems.
- Review Task overview: Managing HTTP sessions in the *Administering applications and their environment* PDF book for information on how to configure the session manager, and best practices for using it.
- Check to see if the problem has been identified and documented by looking at the available online support (hints and tips, technotes, and fixes).
- If you don't find your problem listed there contact IBM support.

HTTP sessions are not getting created, or are lost between requests

By default, the session manager uses cookies to store the session ID on the client between requests. Unless you intend to avoid cookie-based session tracking, ensure that cookies are flowing between WebSphere Application Server and the browser:
- Make sure the **Enable cookies** check box is checked under the **Session tracking Mechanism** property.
- Make sure cookies are enabled on the browser you are testing from or from which your users are accessing the application.
- Check the Cookie domain specified on the SessionManager (to view the or update the cookie settings, in the **Session tracking mechanism->enable cookies** property, click **Modify**).
  - For example, if the cookie domain is set as ″.myCom.com″, resources should be accessed using that domain name. Example: http://www.myCom.com/myapp/servlet/sessionservlet.
  - If the domain property is set, make sure it begins with a dot (.). Certain versions of Netscape do not accept cookies if domain name doesn't start with a dot. Internet Explorer honors the domain with or without a dot. For example, if the domain name is set to *mycom.com*, change it to *.mycom.com* so that both Netscape and Internet Explorer honor the cookie.

    Note: When the servers are on different hosts, ensure that session cookies flow to all the servers by configuring a front-end router such as a Web server with the plug-in or setting the Cookie domain.
- Check the **Cookie path** specified on the SessionManager. Check whether the problem URL is hierarchically below the Cookie path specified. If not correct the Cookie path.
- If the `Cookie maximum age` property is set, ensure that the client (browser) machine's date and time is the same as the server's, including the time zone. If the client and the server time difference is over the ″Cookie maximum age″ then every access would be a new session, since the cookie will ″expire″ after the access.
- If you have multiple Web modules within an enterprise application that track sessions:
  - If you want to have different session settings among Web modules in an enterprise application, ensure that each Web module specifies a different cookie name or path, or

  - If Web modules within an enterprise application use a common cookie name and path, ensure that the HTTP session settings, such as Cookie maximum age, are the same for all Web modules. Otherwise cookie behavior will be unpredictable, and will depend upon which application creates the session. Note that this does not affect session data, which is maintained separately by Web module.
- Check the cookie flow between browser and server:
  1. On the browser, enable ″cookie prompt″. Hit the servlet and make sure cookie is being prompted.
  2. On the server, enable SessionManager trace. Enable tracing for the HTTP session manager component, by using the trace specification ″com.ibm.ws.webcontainer.httpsession.*=all=enabled″. After trace is enabled, exercise your session-using servlet or jsp, then follow the instructions for dumping and browsing the trace output .
  3. Access the session servlet from the browser.
  4. The browser will prompt for the cookie; note the jsessionid.
  5. Reload the servlet, note down the cookie if a new cookie is sent.
  6. Check the session trace and look for the session id and trace the request by the thread. Verify that the session is stable across Web requests:
     - Look for **getIHttpsession(...)** which is start of session request.
     - Look for **releaseSesson(..)** which is end of servlet request.
- If you are using URL rewriting instead of cookies:
  - Ensure there are no static HTML pages on your application's navigation path.
  - Ensure that your servlets and JSP files are implementing URL rewriting correctly. For details and an example see the Session tracking options section in the *Developing and deploying applications* PDF book.
- If you are using SSL as your session tracking mechanism:
  - Ensure that you have SSL enabled on your IBM HTTP Server or iPlanet HTTP server.
  - Review the Session tracking options section in the *Developing and deploying applications* PDF book..
- If you are in a clustered (multiple node) environment, ensure that you have session persistence enabled.

HTTP Sessions are not persistent

If your HTTP sessions are not persistent, that is session data is lost when the application server restarts or is not shared across the cluster:
- Check the data source.
- Check the session manager's persistence settings properties:
  - If you intend to take advantage of session persistence, verify that Persistence is set to **Database**.
  - Persistence could also be set to **Memory-to-Memory Replication**.
  - If you are using **Database-based persistence**:
    - Check the JNDI name of the data source specified correctly on SessionManager.
    - Specify correct userid and password for accessing the database.

      Note that these settings have to be checked against the properties of an existing data source in the administrative console. The session manager does not automatically create a session database for you.
    - The data source should be non-JTA, for example, non XA enabled.
    - Check the JVM logs for appropriate database error messages.
    - With DB2, for row sizes other than 4k make sure specified row size matches the DB2 page size. Make sure tablespace name is specified correctly.

Session is shared across multiple browsers on same client machine

This behavior is browser-dependent. It varies between browser vendors, and also may change according to whether a browser is launched as a new process or as a subprocess of an existing browser session (for example by hitting Ctl-N on Windows).

The Cookie maximum age property of the session manager also affects this behavior, if cookies are used as the session-tracking mechanism. If the maximum age is set to some positive value, all browser

instances share the cookies, which are persisted to file on the client for the specified maximum age time.

Session is not getting invalidated immediately after specified session timeout interval

The SessionManager invalidation process thread runs every *x* seconds to invalidate any invalid sessions, where x is determined based on the session timeout interval specified in the session manager properties. For the default value of 30 minutes , *x* is around 300 seconds. In this case, it could take up to 5 minutes (300 seconds) beyond the timeout threshold of 30 minutes for a particular session to become invalidated.

Unwanted sessions are being created by JavaServer Pages

As required by the JavaServer Pages (JSP) specification, JSP pages by default perform a request.getSession(true), so that a session is created if none exists for the client. To prevent JSP pages from creating a new session, set the session scope to **false** in the .jsp file using the page directive as follows:

```
<% @page session="false" %>
```

Session data intended for one client is seen by another client

In rare situations, usually due to application errors, session data intended for one client might be seen by another client. This situation is referred to as session data crossover. When the *DebugSessionCrossover* custom property is set to true, code is enabled to detect and log instances of session data crossover. Checks are performed to verify that only the session associated with the request is accessed or referenced. Messages are logged if any discrepancies are detected. These messages provide a starting point for debugging this problem. This additional checking is only performed when running on the WebSphere-managed dispatch thread, not on any user-created threads.

For additional information on how to set this property, see article, Web container custom properties in the *Administering applications and their environment* PDF book.

A ClassCastException error occurs during failover of a session that contains an Enterprise JavaBeans (EJB) reference

If you run WebSphere® Application Server for z/OS® Version 6.0.1 and configure a session manager to replicate EJB references, a session failover might trigger display of the following exception in the server region job log:

```
java.lang.ClassCastException: cannot cast class
       org.omg.stub.java.rmi._Remote_Stub to interface javax.ejb.EJBObject
```

The log also displays a null pointer exception. The problem results from the session outbound request, where WebSphere Application Server for z/OS issued a CORBA::COMM_FAILURE exception with a C9C21355 minor code. This behavior occurs because your application server contains all of the following configurations:

1. SAF is both the local operating system, as well as the user registry
2. Attribute propagation is enabled
3. An unauthenticated user initiated the session outbound request

To correct this problem apply the APAR PK06777 fix to WebSphere Application Server for z/OS V6.0.1. You can retain the previously mentioned server configurations.

IBM Support has documents and tools that can save you time gathering information needed to resolve problems as described in “Troubleshooting help from IBM” on page 126. Before opening a problem report, see the Support page:

- http://www.ibm.com/servers/eserver/support/iseries/software/v5r3/index.html

Chapter 8. SIP applications

Tracing a SIP container

You can trace a Session Initiation Protocol (SIP) container, starting either immediately or after the next server startup. This tracing writes a record of SIP events to a log file.

Follow these steps to start tracing a SIP container:

1. Open the administrative console. For more information about the console, read the Using the administrative console chapter of the *Administering applications and their environment* PDF book.
2. In the administrative console, click **Troubleshooting** → **Logs and trace**.
3. Select the name of the server for the SIP container.
4. From the General Properties section, click **Diagnostic Trace Service**.
5. Under the Additional Properties section, click **Change Log Detail Levels**
6. Select one of the following options:

Option	Description
Configuration	To start tracing after the next server startup
Runtime	To start tracing immediately

7. Replace the content of the trace specification with the following code: `com.ibm.ws.sip.*=all=enabled`.

   Note: If you want monitor only specific pieces of SIP containers, expand the **com.ibm.ws.sip** section and select the individual items you wish to trace.
8. Make sure that the **Enable trace with following specification** check box is checked.
9. Click **Apply** → **Save**.

When the changes take effect (refer to step 6 above), SIP-level tracing messages appear in *WASProductDir*/logs/*serverName*/trace.log, where *WASProductDir* is the fully qualified path name of the directory in which the product is installed and *serverName* is the name of the specific instance of the application server that is running the SIP container to be traced. These messages include application load events as well as SIP request and response parsing and SIP servlet invocation.

Chapter 9. EJB applications

Access intent exceptions

The exceptions thrown in response to the application of access intent policies are listed.

com.ibm.ws.ejbpersistence.utilpm.PersistenceManagerException

If the method that drives the ejbLoad() method is configured to be read-only but updates are then made within the transaction that loaded the bean's state, an exception is thrown during invocation of the ejbStore() method, and the transaction is rolled back. Likewise, the ejbRemove() method cannot succeed in a transaction that is set as read-only. If an update hint is applied to methods of entity beans with bean-managed persistence, the same behavior and exception results. The forwarded exception object contains the message string `PMGR1103E: update instance level read only bean beanName`

This exception is also thrown if the applied access intent policy cannot be honored because a finder, ejbSelect, or container-managed relationship (CMR) accessor method returns an inherently read-only result. The forwarded exception object contains the message string `PMGR1001: No such DataAccessSpec - methodName`

The most common occurrence of this error is when a custom finder that contains a read-only EJB Query Language (EJB QL) statement is called with an applied access intent of `wsPessimisticUpdate` or `wsPessimisticUpdate-Exclusive`. These policies require the use of a USE AND KEEP UPDATE LOCKS clause on the SQL SELECT statement to be executed, but a read-only query cannot support USE AND KEEP UPDATE LOCKS. Other examples of read-only queries include joins; the use of ORDER BY, GROUP BY, and DISTINCT keywords.

To eliminate the exception, edit the EJB query so that it does not return an inherently read-only result or change the access intent policy being applied.

- If an update access is required, change the applied access intent setting to `wsPessimisticUpdate-WeakestLockAtLoad` or `wsOptimisticUpdate`.
- If update access is not truly required, use `wsPessimisticRead` or `wsOptimisticRead`.
- If connection sharing between entity beans is required, use `wsPessimisticUpdate-WeakestLockAtLoad` or `wsPessimisticRead`.

com.ibm.websphere.ejb.container.CollectionCannotBeFurtherAccessed

If a lazy collection is driven after it is no longer in scope, and beyond what has already been locally buffered, a CollectionCannotBeFurtherAccessed exception is thrown.

com.ibm.ws.exception.RuntimeWarning

If an application is configured incorrectly, a run-time warning exception is thrown as the application starts; startup is ended. You can validate an application's configuration by choosing the verify function. Some examples of misconfiguration include:

- A method configured with two different access intent policies
- A method configured with an undefined access intent policy

Frequently asked questions: Access intent

The following frequently asked questions involving access intent are answered.

I have not applied any access intent policies at all. My application runs just fine with a DB2 database, but it fails with an Oracle database with the following message: *com.ibm.ws.ejbpersistence.utilpm.PersistenceManagerException: PMGR1001E: No such DataAccessSpec :FindAllCustomers. The backend datastore does not support the SQLStatement needed by this AccessIntent: (pessimistic update-weakestLockAtLoad)(collections: transaction/25) (resource manager prefetch: 0) (AccessIntentImpl@d23690a)*. Why?

If you have not configured access intent, all of your data is accessed under the default access intent policy (`wsPessimisticUpdate-WeakestLockAtLoad`). On DB2 the weakest lock is share. On Oracle databases,

however, the weakest lock is update; this means that the SQL query must contain a FOR UPDATE clause. To avoid this problem, try to apply an access intent policy that supports optimistic concurrency.

I am calling a finder method and I get an InconsistentAccessIntentException at run time. Why?

This can occur when you use method-level access intent policies to apply more control over how a bean instance is loaded. This exception indicates that the entity bean was previously loaded in the same transaction. This could happen if you called a multifinder method that returned the bean instance with access intent policy X applied; you are now trying to load the second bean again by calling its findByPrimaryKey method with access intent Y applied. Both methods must have the same access intent policy applied.

Likewise, if the entity was loaded once in the transaction using an access intent policy configured on a finder, you might have called a container-managed relationship (CMR) accessor method that returned the entity bean configured to load using that entity's default access intent.

To avoid this problem, ensure that your code does not load the same bean instance twice within the same transaction with different access intent policies applied. Avoid the use of method-level access intent unless absolutely necessary.

I have two beans in a container-managed relationship. I call findByPrimaryKey() on the first bean and then call getBean2( ), a CMR accessor method, on the returned instance. At that point, I get an InconsistentAccessIntentException. Why?

You are probably using read-ahead. When you loaded the first bean, you caused the second bean to be loaded under the access intent policy applied to the finder method for the first bean. However, you have configured your CMR accessor method from the first bean to the second with a different access intent policy. CMR accessor methods are really finder methods in disguise; the run-time environment behaves as if you were trying to change the access intent for an instance you have already read from persistent store.

To avoid this problem, beans configured in a read-ahead hint are all driven to load with the same access intent policy as the bean to which the read-ahead hint is applied.

I have a bean with a one-to-many relationship to a second bean. The first bean has a pessimistic-update intent policy applied. When I try to add an instance of the second bean to the first bean's collection, I get an UpdateCannotProceedWithIntegrityException. Why?

The second bean probably has a read intent policy applied. When you add the second bean to the first bean's collection, you are not updating the first bean's state, you are implicitly modifying the second bean's state. (The second bean contains a foreign key to the first bean, which is modified.)

To avoid this problem, ensure that both ends of the relationship have an update intent policy applied if you expect to change the relationship at run time.

Troubleshooting tips for EJBDEPLOY relationships

This article provides troubleshooting information for EJBDEPLOY problems.

The converter that is defined for the primary key is not invoked on its foreign key value

The mapping for primary key fields to database columns may use a converter to transform the key values. If a container-managed persistence (CMP) bean uses a converter to map its primary key, and that bean has a relationship where the bean at the other end holds a foreign key, the mapping for the foreign key will not use the converter.

The following errors might occur, indicating that the converter defined for the primary key is not invoked on its foreign key value. During the run of the ejbDeploy command , you receive the following message:

```
No type mapping defined for Java datatype1 to Database datatype2
```

During run time, the application does not find the CMP bean at the other end of the relationship.

To work around this limitation, define your own foreign key in the database table, and create a mapping that uses the same converter as defined for the primary key on the enterprise beans at the other end of its relationship.

Enterprise bean and EJB container troubleshooting tips

If you are having problems starting an EJB container, or encounter error messages or exceptions that appear to be generated on by an EJB container, follow these steps to resolve the problem:

- Use the Administrative Console to verify that the application server which hosts the container is running.
- Browse the JVM log files for the application server which hosts the container. Look for the message **server *server_name* open for e-business** in the `SystemOut.log` . If it does not appear, or if you see the message **problems occurred during startup**, browse the `SystemErr.log` for details.
- Browse the system log files for the application server which hosts the container.
- Enable tracing for the EJB Container component, by using the following trace specification `EJBContainer=all=enabled`. Follow the instructions for dumping and browsing the trace output to narrow the origin of the problem.

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

For current information available from IBM Support on known problems and their resolution, see the IBM Support page.

IBM Support has documents that can save you time gathering information needed to resolve this problem. Before opening a PMR, see the IBM Support page.

Error in client log: Missing jar file

The following error message appears in the client log file because a JAR file is missing from the classpath on the client machine. The Object Request Broker (ORB) needs this file to unmarshal the nested exception that is part of the EJB exception, returned by the server to the client application. For example, if the EJB returns a DB2® JCC SQL exception nested inside of the EJB exception that it returns to the client, the ORB is not able to unmarshal the nested exception if the db2jcc.jar file that contains the DB2 SQL exception is not in the client classpath.

```
java.rmi.MarshalException: CORBA MARSHAL 0x4942f89a No; nested exception is:
org.omg.CORBA.MARSHAL: Unable to read value
from underlying bridge : Custom marshaling (4) Sender's class does not match
local class  vmcid: 0x4942f000  minor  code: 2202  completed: No*
```

To avoid this error, include the JAR file that contains the class for the nested exception that is returned in the EJB exception.

Cannot access an enterprise bean from a servlet, a JSP file, a stand-alone program, or another client

This article provides troubleshooting tips for problems related to accessing enterprise beans.

What kind of error are you seeing?
- **javax.naming.NameNotFoundException: Name *name* not found in context ″local″ message** when access is attempted
- **BeanNotReentrantException** is thrown
- **CSITransactionRolledbackException / TransactionRolledbackException** is thrown
- Call fails, Stack trace beginning **EJSContainer E Bean method threw exception [exception_name]** found in JVM log file.
- Call fails, **ObjectNotFoundException or ObjectNotFoundLocalException** when accessing stateful session EJB found in JVM log file.
- Attempt to start CMP EJB module fails with **javax.naming.NameNotFoundException: *dataSourceName***
- **Transaction [tran ID] has timed out after 120 seconds** error accessing EJB.
-
- Symptom: **CNTR0001W: A Stateful SessionBean could not be passivated**
- Symptom: **org.omg.CORBA.BAD_PARAM: Servant is not of the expected type. minor code: 4942F21E completed: No** returned to client program when attempting to execute an EJB method

If the client is remote to the enterprise bean, which means, running in a different application server or as a stand-alone client, browse the JVM logs of the application server hosting the enterprise bean as well as log files of the client.

If you do not see a problem that resembles yours, or if the information provided does not solve your problem, perform these steps:
1. If the problem appears to be name-service related, which means that you see a NameNotFoundException, or a message ID beginning with NMSV, see these topics for more information:
   - “Cannot look up an object hosted by WebSphere Application Server from a servlet, JSP file, or other client” on page 364
   - “Naming service troubleshooting tips” on page 363
2. Check to see if the problem is identified and documented using the links in Diagnosing and fixing problems: Resources for learning.

If you still cannot fix your problem, see“Troubleshooting help from IBM” on page 126 for further assistance.

ObjectNotFoundException or ObjectNotFoundLocalException when accessing stateful session EJB

A possible cause of this problem is that the stateful session bean timed out and was removed by the container. This event must be addressed in the code, according to the EJB 2.1 specification (available at `http://java.sun.com/products/ejb/docs.html`), section 7.6.2, Dealing with exceptions.

Stack trace beginning ″EJSContainer E Bean method threw exception [exception_name]″ found in JVM log file

If the exception name indicates an exception thrown by an IBM class that begins with ″com.ibm...″, then search for the exception name within the information center, and in the online help as described below. If ″exception name″ indicates an exception thrown by your application, contact the application developer to determine the cause.

javax.naming.NameNotFoundException: Name name not found in context ″local″

A possible reason for this exception is that the enterprise bean is not local (not running in the same Java virtual machine [JVM] or application server) to the client JSP, servlet, Java application, or other enterprise bean, yet the call is to a ″local″ interface method of the enterprise bean . If access worked in a development environment but not when deployed to WebSphere Application Server, for example, it might be that the enterprise bean and its client were in the same JVM in development, but are in separate processes after deployment.

To resolve this problem, contact the developer of the enterprise bean and determine whether the client call is to a method in the local interface for the enterprise bean. If so, have the client code changed to call a remote interface method, or to promote the local method into the remote interface.

References to enterprise beans with local interfaces are bound in a name space local to the server process with the URL scheme of `local:`. To obtain a dump of a server `local:` name space, use the name space dump utility described in the article ″“Name space dump utility for `java:`, `local:` and `server` name spaces” on page 370.″

BeanNotReentrantException is thrown

This problem can occur because client code (typically a servlet or JSP file) is attempting to call the same stateful SessionBean from two different client threads. This situation often results when an application stores the reference to the stateful session bean in a static variable, uses a global (static) JSP variable to refer to the stateful SessionBean reference, or stores the stateful SessionBean reference in the HTTP session object. The application then has the client browser issue a new request to the servlet or JSP file before the previous request has completed.

To resolve this problem, ask the developer of the client code to review the code for these conditions.

CSITransactionRolledbackException / TransactionRolledbackException is thrown

An enterprise bean container creates these high-level exceptions to indicate that an enterprise bean call could not successfully complete. When this exception is thrown, browse the JVM logs to determine the underlying cause.

Some possible causes include:

- The enterprise bean might throw an exception that was not declared as part of its method signature. The container is required to roll back the transaction in this case. Common causes of this situation are where the enterprise bean or code that it calls creates a NullPointerException, ArrayIndexOutOfBoundsException, or other Java runtime exception, or where a BMP bean encounters a JDBC error. The resolution is to investigate the enterprise bean code and resolve the underlying exception, or to add the exception to the problem method signature.
- A transaction might attempt to do additional work after being placed in a `″Marked Rollback″`, `″RollingBack″`, or `″RolledBack″` state. Transactions cannot continue to do work after they are set to one of these states. This situation occurs because the transaction has timed out which, often occurs because of a database deadlock. Work with the application database management tools or administrator to determine whether database transactions called by the enterprise bean are timing out.
- A transaction might fail on commit due to dangling work from local transactions. The local transaction encounters some ″dangling work″ during commit. When a local transactions encounters an ″unresolved action″ the default action is to ″rollback″. You can adjust this action to ″commit″ in an assembly tool. Open the enterprise bean `.jar` file (or the `EAR` file containing the enterprise bean) and select the Session Beans or Entity Beans object in the component tree on the left. The Unresolved Action property is on the IBM Extensions tab of the container properties.

Attempt to start EJB module fails with ″javax.naming.NameNotFoundException dataSourceName_CMP″exception

This problem can occur because:
- When the DataSource resource was configured, container managed persistence was not selected.
  - To confirm this problem, in the administrative console, browse the properties of the data source given in the NameNotFoundException. On the Configuration panel, look for a check box labeled **Container Managed Persistence**.
  - To correct this problem, select the check box for **Container Managed Persistence**.
- If container managed persistence is selected, it is possible that the CMP DataSource could not be bound into the namespace.
  - Look for additional naming warnings or errors in the status bar, and in the hosting application server JVM logs. Check any further naming-exception problems that you find by looking at the topic “Cannot look up an object hosted by WebSphere Application Server from a servlet, JSP file, or other client” on page 364.

Transaction [tran ID] has timed out after 120 seconds accessing an enterprise bean

This error can occur when a client executes a transaction on a CMP or BMP enterprise bean.
- The default timeout value for enterprise bean transactions is 120 seconds. After this time, the transaction times out and the connection closes.
- If the transaction legitimately takes longer than the specified timeout period, go to **Manage Application Servers > *server_name***, select the **Transaction Service properties** page, and look at the property **Total transaction lifetime timeout**. Increase this value if necessary and save the configuration.

Symptom:CNTR0001W: A Stateful SessionBean could not be passivated

This error can occur when a Connection object used in the bean is not closed or nulled out.

To confirm this is the problem, look for an exception stack in the JVM log for the EJB container that hosts the enterprise bean, and looks similar to:

```
[time EDT] <ThreadID> StatefulPassi W CNTR0001W:
A Stateful SessionBean could not be passivated: StatefulBean0
(BeanId(XXX#YYY.jar#ZZZZ, <ThreadID>),
state = PASSIVATING) com.ibm.ejs.container.passivator.StatefulPassivator@<ThreadID>
java.io.NotSerializableException: com.ibm.ws.rsadapter.jdbc.WSJdbcConnection
 at java.io.ObjectOutputStream.outputObject((Compiled Code))
 at java.io.ObjectOutputStream.writeObject(ObjectOutputStream.java(Compiled Code))
 at java.io.ObjectOutputStream.outputClassFields((Compiled Code))
 at java.io.ObjectOutputStream.defaultWriteObject((Compiled Code))
 at java.io.ObjectOutputStream.outputObject((Compiled Code))
 at java.io.ObjectOutputStream.writeObject(ObjectOutputStream.java(Compiled Code))
 at com.ibm.ejs.container.passivator.StatefulPassivator.passivate((Compiled Code))

 at com.ibm.ejs.container.StatefulBean0.passivate((Compiled Code)
 at com.ibm.ejs.container.activator.StatefulASActivationStrategy.atUnitOfWorkEnd
                     ((Compiled Code))
 at com.ibm.ejs.container.activator.Activator.unitOfWorkEnd((Compiled Code))
 at com.ibm.ejs.container.ContainerAS.afterCompletion((Compiled Code)
```

where *XXX,YYY,ZZZ* is the Bean’s name, and *<ThreadID>* is the thread ID for that run.

To correct this problem, the application must close all connections and set the reference to null for all connections. Typically this activity is done in the `ejbPassivate()` method of the bean. See the enterprise bean specification mandating this requirement, specifically section 7.4 in the EJB specification Version 2.1. Also, note that the bean must have code to reacquire these connections when the bean is reactivated. Otherwise, there are NullPointerExceptions when the application tries to reuse the connections.

Symptom: org.omg.CORBA.BAD_PARAM: Servant is not of the expected type. minor code: 4942F21E completed: No

This error can be returned to a client program when the program attempts to execute an EJB method.

Typically this problem is caused by a mismatch between the interface definition and implementation of the client and server installations, respectively.

Another possible cause is when an application server is set up to use a single class loading scheme. If an application is uninstalled while the application server remains active, the classes of the uninstalled application are still loaded in the application server. If you change the application, redeploy and reinstall it on the application server, the previously loaded classes become back level. The back level classes cause a code version mismatch between the client and the server.

To correct this problem:
1. Change the application server class loading scheme to multiple.
2. Stop and restart the application server and try the operation again.
3. Make sure the client and server code version are the same.

Important file for message-driven beans

The *server_name*-durableSubscriptions.ser file in the WAS_HOME/temp directory is important for the operation of the WebSphere Application Server messaging service, so should not be deleted.

If you do need to delete the WAS_HOME/temp directory or other files in it, ensure that you preserve the following file:

server_name-**durableSubscriptions.ser**

You should not delete this file, because the messaging service uses it to keep track of durable subscriptions for message-driven beans. If you uninstall an application that contains a message-driven bean, this file is used to unsubscribe the durable subscription.

Chapter 10. Client applications

A client program does not work

What kind of problem are you seeing?

ActiveX client fails to display ASP files, or WebSphere Application Server resources (JSP files, servlet, or HTML pages) or both

A possible cause of this problem is that both IIS for serving Active Server Pages (ASP) files and an HTTP server that supports WebSphere Application Server (such as IBM HTTP Server) are deployed on the same host. This deployment leads to misdirected HTTP traffic if both servers are listening on the same port (such as the default port 80).

To resolve this problem, either:
- Open the IIS administrative panel, and edit the properties of the default Web server to change the port number to a value other than 80
- Install IIS and the HTTP server on separate servers.

For current information available from IBM Support on known problems and their resolution, see the IBM Support page.

IBM Support has documents that can save you time gathering information needed to resolve this problem. Before opening a PMR, see the IBM Support page.

Application client troubleshooting tips

This topic provides debugging tips for resolving common Java 2 Platform Enterprise Edition (J2EE) application client problems. To use this troubleshooting guide, review the trace entries for one of the J2EE application client exceptions, and then locate the exception in the guide.

Some of the errors in the guide are samples, and the actual error you receive can be different than what is shown here. You might find it useful to rerun the `launchClient` command specifying the `-CCverbose=true` option. This option provides additional information when the J2EE application client run time is initializing.

Error: java.lang.NoClassDefFoundError

Explanation This exception is thrown when Java code cannot load the specified class.

Possible causes
- Invalid or non-existent class
- Class path problem
- Manifest problem

Recommended response

Check to determine if the specified class exists in a Java Archive (JAR) file within your Enterprise Archive (EAR) file. If it does, make sure the path for the class is correct. For example, if you get the exception:

```
java.lang.NoClassDefFoundError:
WebSphereSamples.HelloEJB.HelloHome
```

verify that the HelloHome class exists in one of the JAR files in your EAR file. If it exists, verify that the path for the class is WebSphereSamples.HelloEJB.

If both the class and path are correct, then it is a class path issue. Most likely, you do not have the failing class JAR file specified in the client JAR file manifest. To verify this situation, perform the following steps:

1. Open your EAR file with the Application Server Toolkit or the Rational Web Developer assembly tool, and select the Application Client.
2. Add the names of the other JAR files in the EAR file to the Classpath field.

This exception is generally caused by a missing Enterprise Java Beans (EJB) module name from the Classpath field.

If you have multiple JAR files to enter in the Classpath field, be sure to separate the JAR names with spaces.

If you still have the problem, you have a situation where a class is loaded from the file system instead of the EAR file. This error is difficult to debug because the offending class is not the one specified in the exception. Instead, another class is loaded from the file system before the one specified in the exception. To correct this error, review the class paths specified with the -CCclasspath option and the class paths configured with the Application Client Resource Configuration Tool. Look for classes that also exist in the EAR file. You must resolve the situation where one of the classes is found on the file system instead of in the `.ear` file. Remove entries from the classpaths, or include the `.jar` files and classes in the `.ear` file instead of referencing them from the file system.

If you use the -CCclasspath parameter or resource classpaths in the Application Client Resource Configuration Tool, and you have configured multiple JAR files or classes, verify they are separated with the correct character for your operating system. Unlike the Classpath field, these class path fields use platform-specific separator characters, usually a colon (on operating systems such as AIX or Linux) or a semi-colon (on Windows systems).
Note: The system class path is not used by the Application Client run time if you use the launchClient batch or shell files. In this case, the system class path would not cause this problem. However, if you load the launchClient class directly, you do have to search through the system class path as well.

Error: com.ibm.websphere.naming.CannotInstantiateObjectException: Exception occurred while attempting to get an instance of the object for the specified reference object. [Root exception is javax.naming.NameNotFoundException: xxxxxxxxxx]

Explanation

This exception occurs when you perform a lookup on an object that is not installed on the host server. Your program can look up the name in the local client Java Naming and Directory Interface (JNDI) name space, but received a NameNotFoundException exception because it is not located on the host server. One typical example is looking up an EJB component that is not installed on the host server that you access. This exception might also occur if the JNDI name you configured in your Application Client module does not match the actual JNDI name of the resource on the host server.

Possible causes

- Incorrect host server invoked
- Resource is not defined
- Resource is not installed
- Application server is not started
- Invalid JNDI configuration

Recommended response

If you are accessing the wrong host server, run the `launchClient` command again with the -CCBootstrapHost parameter specifying the correct host server name. If you are accessing the correct host server, use the product `dumpnamespace` command line tool to see a listing of the host server JNDI name space. If you do not see the failing object name, the resource is either not installed on the host server or the appropriate application server is not started. If you determine the resource is already installed and started, your JNDI name in your client application does not match the global JNDI name on the host server. Use the Application Server Toolkit to compare the JNDI bindings value of the failing object name in the client application to the JNDI bindings value of the object in the host server application. The values must match.

Error: javax.naming.ServiceUnavailableException: A communication failure occurred while attempting to obtain an initial context using the provider url: ″iiop://[invalidhostname]″. Make sure that the host and port information is correct and that the server identified by the provider URL is a running name server. If no port number is specified, the default port number 2809 is used. Other possible causes include the network environment or workstation network configuration. Root exception is org.omg.CORBA.INTERNAL: JORB0050E: In Profile.getIPAddress(), InetAddress.getByName[invalidhostname] threw an UnknownHostException. minor code: 4942F5B6 completed: Maybe

Explanation

This exception occurs when you specify an invalid host server name.

Possible causes

- Incorrect host server invoked
- Invalid host server name

Recommended response

Run the `launchClient` command again and specify the correct name of your host server with the -CCBootstrapHost parameter.

Error: javax.naming.CommunicationException: Could not obtain an initial context due to a communication failure. Since no provider URL was specified, either the bootrap host and port of an existing ORB was used, or a new ORB instance was created and initialized with the default bootstrap host of ″localhost″ and the default bootstrap port of 2809. Make sure the ORB bootstrap host and port resolve to a running name server. Root exception is org.omg.CORBA.COMM_FAILURE: WRITE_ERROR_SEND_1 minor code: 49421050 completed: No

Explanation

This exception occurs when you run the `launchClient` command to a host server that does not have the Application Server started. You also receive this exception when you specify an invalid host server name. This situation might occur if you do not specify a host server name when you run the launchClient tool. The default behavior is for the launchClient tool to run to the local host, because WebSphere Application Server does not know the name of your host server. This default behavior only works when you are running the client on the same machine with WebSphere Application Server is installed.

Possible causes

- Incorrect host server invoked
- Invalid host server name
- Invalid reference to `localhost`
- Application server is not started
- Invalid bootstrap port

Recommended response

If you are not running to the correct host server, run the `launchClient` command again and specify the name of your host server with the `-CCBootstrapHost` parameter. Otherwise, start the Application Server on the host server and run the `launchClient` command again.

Error: javax.naming.NameNotFoundException: Name comp/env/ejb not found in context ″java:″

Explanation

This exception is thrown when the Java code cannot locate the specified name in the local JNDI name space.

Possible causes

- No binding information for the specified name
- Binding information for the specified name is incorrect
- Wrong class loader was used to load one of the program classes
- A resource reference does not include any client configuration information
- A client container on the deployment manager is trying to use enterprise extensions (not supported)

Recommended response

Open the EAR file with the Application Server Toolkit, and check the bindings for the failing name. Ensure this information is correct. If you are using Resource References, open the EAR file with the Application Client Resource Configuration Tool, and verify that the Resource Reference has client configuration information and the name of the Resource Reference exactly matches the JNDI name of the client configuration. If the values are correct, you might have a class loader error. For detailed information about the configuration tool and opening EAR files, read the Starting the Application Client Resource Configuration Tool in the *Developing and deploying applications* PDF book.

Error: java.lang.ClassCastException: Unable to load class: org.omg.stub.WebSphereSamples.HelloEJB._HelloHome_Stub at com.ibm.rmi.javax.rmi.PortableRemoteObject.narrow(portableRemoteObject.java:269)

Explanation

This exception occurs when the application program attempts to narrow to the EJB home class and the class loaders cannot find the EJB client side bindings.

Possible causes

- The files, *_Stub.class and _Tie.class, are not in the EJB `.jar` file
- Class loader could not find the classes

Recommended response

Look at the EJB `.jar` file located in the `.ear` file and verify the class contains the Enterprise Java Beans (EJB) client side bindings. These are class files with file names that end in `_Stub` and `_Tie`. If the binding classes are in the EJB `.jar` file, then you might have a class loader error.

Error: WSCL0210E: The Enterprise archive file [EAR file name] could not be found. com.ibm.websphere.client.applicationclient.ClientContainerException: com.ibm.etools.archive.exception.OpenFailureException

Explanation

This error occurs when the application client run time cannot read the Enterprise Archive (EAR) file.

Possible causes

The most likely cause of this error is that the system cannot find the EAR file cannot be found in the path specified on the `launchClient` command.

Recommended response

Verify that the path and file name specified on the `launchclient` command are correct. If you are running on the Windows operating system and the path and file name are correct, use a short version of the path and file name (8 character file name and 3 character extension).

The launchClient command appears to hang and does not return to the command line when the client application has finished.

Explanation

When running your application client using the `launchClient` command the WebSphere Application Server run time might need to display the security login dialog. To display this dialog, WebSphere Application Server run time creates an Abstract Window Toolkit (AWT) thread. When your application returns from its main method to the application client run time, the application client run time attempts to return to the operating system and end the Java virtual machine (JVM) code. However, since there is an AWT thread, the JVM code will not end until `System.exit` is called.

Possible causes

The JVM code does not end because there is an AWT thread. Java code requires that `System.exit()` be called to end AWT threads.

Recommended response

- Modify your application to call `System.exit(0)` as the last statement.
- Use the `-CCexitVM=true` parameter when you call the `launchClient` command.

The applet client application client fails to launch an HTML browser in Internet Explorer

Explanation

Applet client applications run only on Windows systems. When the applet client application runs, the application output data is displayed in a browser window. If you are using Internet Explorer with the Windows XP operating system for Service Pack 2 , then you might get errors when trying to display output data.

Possible causes

The Windows XP operating system for Service Pack 2 has a security feature that blocks pop-up browser windows from appearing.

Recommended response

- Locate the information bar found under the URL Address bar in the Internet Explorer pop-up browser that has been blocked.
- Click the Information Bar to display options that disable the operating system security feature.
- Select **Allow blocked content**. You are prompted with a security window asking you to confirm your selection to allow blocked content.
- Click **Yes**.
- The applet client application runs successfully, and the browser information is displayed appropriately.

Installing the Developer Kit feature downgrades the JRE files from Version 6.0.1 or Version 6.0.2 to Version 6.0

Explanation

If you select the Developer Kit feature on the Application Client Version 6.0.0 installer, all the files are installed under the <*client_install_root*>/java directory, instead of leaving the Java Runtime Environment (JRE) files intact. The JRE files are unexpectedly downgraded to the Version 6.0.0 level.

Possible causes

Selecting the **Developer Kit** feature on the Application Client 6.0.0 installer will actually install all files under the <*client_install_root*>/java directory rather than leave the JRE files intact. Therefore, the JRE files are unexpected downgraded to the 6.0.0 level in the above installation scenario.

Recommended response

Complete the following installation steps to prevent unexpected downgrading of the JRE files.

IBM Support has documents and tools that can save you time gathering information needed to resolve problems as described in “Troubleshooting help from IBM” on page 126. Before opening a problem report, see the Support page:

- http://www.ibm.com/servers/eserver/support/iseries/software/v5r3/index.html

Chapter 11. Web services

Troubleshooting Web services

This topic provides an entry into the topics available to learn about ways that you can troubleshoot Web services applications.

This topic provides information for you to troubleshoot during different steps of the development, assembly, deployment, and security processes of a Web service.

Select the Web services topic area that you want to troubleshoot:

- Command-line tools

  This topic provides information on troubleshooting the **WSDL2Java** command-line tool and the **Java2WSDL** command-line tool.
- Java compiler errors

  This topic discusses troubleshooting compiled bindings of Web services.
- Serialization or deserialization errors

  This topic presents problems you might encounter performing serialization and deserialization in Web services.
- Authentication challenges and authorization failures with Web services security

  This topic discusses troubleshooting authentication and authorization when you are securing Web services.

Troubleshooting Web services command-line tools

This topic discusses troubleshooting the **WSDL2Java** and **Java2WSDL** command-line tools that are used when you develop Web services.

Each section in this topic is a problem that you might experience while using the WSDL2Java or Java2WSDL tool. A solution is provided to help you troubleshoot the problem.

The .Net client does not reflect a Web service method with Vector-type parameters

The following exception displays when running the .NET client for a Web service:

```
System.InvalidOperationException: Method AnnuityInteropService.wsListAnnuityByHolder can not be reflected.
System.InvalidOperationException: There was an error reflecting wsListAnnuityByHolderResult.
System.InvalidOperationException: The Form property might not be unqualified when an explicit namespace
property is available.
```

The problem is exposed when the server-side method returns a Vector and the style of the Web Services Description Language (WSDL) file is document/literal and the Form is unqualified. The unqualified Form is always generated for the document/literal style because elementFormDefault=″unqualified″ by default.

A problem exists in the .NET framework that can generate a Web service proxy class that is not valid when your Web service methods contain certain arrays as parameters. The framework adds an XmlArrayAttribute attribute to the parameter, and the attribute constructor contains an ″Form=Unqualified″ and a namespace definition. While the attribute is valid, it is not valid to combine these two aspects of the attribute, causing the `System.InvalidOperationException` exception.

To avoid this problem, do not use Vector parameters in a Web service method that is deployed into WebSphere Application Server. Vector parameters in a Web service method do not work with a .Net client or a Java collection type.

Error with the Endpoint Enabler tool on a Hyper Threading-enabled machine

When you run the Endpoint Enabler tool on a Hyper Threading-enabled machine, an error is reported in the Tasks view against the Enterprise Application project. The Enterprise Application project cannot be deleted until you restart the workbench.

This problem can occur if a race condition exists between the automatic build and the Enterprise JavaBeans (EJB) project validator.

To resolve the problem, turn off the workbench automatic build before running the Endpoint Enabler tool, and turn it back on afterwards.

Multiprotocol port component restrictions with JSR109 Version 1.0 and 1.1

Java Specification Requests (JSR) 109 specification validation errors occur when deploying an enterprise archive (EAR) file that contains a WSDL file with `http`, `jms` and `ejb` bindings generated by the **Java2WSDL** command-line tool.

The JSR 109 specification requires each port component defined in the `webservices.xml` deployment descriptor file to refer to unique <servlet-class> elements in `web.xml` file for a JavaBeans implementation, or a unique <session> element in `ejb-jar.xml` file for an EJB implementation. The servlet and session EJB are located in the `webservices.xml` file represented by <servlet-link> or <ejb-link> element.

The **WSDL2Java** command-line tool maps the ports found in a WSDL file to port components that are in the generated `webservices.xml` file. If a single Web service has multiple bindings, in addition to a port for each of these bindings, the `webservices.xml` file contains multiple port components that should all point to the same EJB module(<session>) or JavaBeans (<servlet-class>) implementation. Because of the JSR 109 restrictions, the `webservices.xml` file is not valid and errors can occur during the deployment process.

The following example displays the error:

```
Error in <module> : CHKW6030E: Implementation class <class> referred to by port
components<port1> and <port2>. (JSR109 1.0: 7.1.2).
```

Here is the error with sample data:

```
Error in WebSvcsInSession20EJB.jar : CHKW6030E: Implementation class WSMultiProtocol
referred to by port components WSMultiProtocolJMS and WSMultiProtocolEJB.(JSR109 1.0: 7.1.2).
```

You can work around the restriction by creating multiple <session> EJB definitions within the `ejb-jar.xml` file that all point to the same implementation class, home interface and remote interface. You can still use the same classes, but the `ejb-jar.xml` file <session> definitions that reference the classes and the interfaces must be duplicated.

The following is an example of the `webservices.xml` file. Look for the classes and interfaces:

```
<webservices>
  <webservice-description>
    <webservice-description-name>WSMultiProtocolService</webservice-description-name>
    <wsdl-file>META-INF/wsdl/WSMultiProtocol.wsdl</wsdl-file>
    <jaxrpc-mapping file>META-INF/WSMultiProtocol_mapping.xml</jaxrpc-mapping file>
    <port-component>
      <port-component-name>WSMultiProtocolEjb</port-component-name>
      <wsdl-port>
        <namespaceURI>http://ejb.pli.tc.wssvt.ibm.com</namespaceURI>
        <localpart>WSMultiProtocolEjb</localpart>
      </wsdl-port>
      <service-endpoint-interface>com.ibm.wssvt.tc.pli.ejb.WSMultiProtocol
   </service-endpoint-interface>
      <service-impl-bean>
        <ejb-link>WSMultiProtocol</ejb-link>
      </service-impl-bean>
    </port-component>
    <port-component>
```

```
  <port-component-name>WSMultiProtocolJMS</port-component-name>
      <wsdl-port>
        <namespaceURI>http://ejb.pli.tc.wssvt.ibm.com</namespaceURI>
        <localpart>WSMultiProtocolJMS</localpart>
      </wsdlport>
      <service-endpoint-interface>com.ibm.wssvt.tc.pli.ejb.WSMultiProtocol
   </service-endpoint-interface>
      <service-impl-bean>
        <ejb-link>WSMultiProtocol_2</ejb-link>
      </service-impl_bean>
  </port-component>
        <port-component>
         <port-component-name>WSMultiProtocolJMS</port-component-name>
      <wsdl-port>
        <namespaceURI>http://ejb.pli.tc.wssvt.ibm.com</namespaceURI>
        <localpart>WSMultiProtocolJMS</localpart>
      </wsdlport>
      <service-endpoint-interface>com.ibm.wssvt.tc.pli.ejb.WSMultiProtocol
   </service-endpoint-interface>
      <service-impl-bean>
        <ejb-link>WSMultiProtocol_3</ejb-link>
      </service-impl_bean>
  </port-component>
 </webservice-description>
</webservices>
```

The following is an example of the `ejb-jar.xml` file. Look for the classes and interfaces, and how they are duplicated:

```
<ejb-jar-id="ejb-jar_ID">
 <display-name>WebSvcsInsSession20EJB</display-name>
 <enterprise-beans>
  <session-id="WSMultiProtocol">
   <ejb-name>WSMultiProtocol</ejb-name>
   <home>com.ibm.wssvt.tc.pli.ejb.WSMultiProtocolHome</home>
   <remote>com.ibm.wssvt.tc.pli.ejb.WSMultiProtocol</remote>
   <ejb-class>com.ibm.wssvt.tc.pli.ejb.WSMultiProtocolWebSvcsBean</ejb-class>
   <session-type>Stateless</session-type>
   <transaction-type>Container</transaction-type>
   <ejb-ref-id="EjbRef_1082407586720">
    <description></description>
    <ejb-ref-name>ejb/BeneficiarySession</ejb-ref-name>
    <ejb-ref-type>Session</ejb-ref-type>
    <home>com.ibm.wssvt.tc.pli.ejb.BeneficiarySessionHome</home>
    <remote>com.ibm.wssvt.tc.pli.ejb.BeneficiarySession</remote>
    <ejb-link>PolicySession20EJB.jar#BeneficiarySession</ejb-link>
   </ejb-ref>
   <ejb-ref-id="EjbRef_1082407586790">
    <description></description>
    <ejb-ref-name>ejb/PolicySession</ejb-ref-name>
    <ejb-ref-type>Session</ejb-ref-type>
    <home>com.ibm.wssvt.tc.pli.ejb.PolicySessionHome</home>
    <remote>com.ibm.wssvt.tc.pli.ejb.PolicySession</remote>
    <ejb-link>PolicySession20EJB.jar#PolicySession</ejb-link>
   </ejb-ref>


  <session-id="WSMultiProtocol_2">
   <ejb-name>WSMultiProtocol_2</ejb-name>
   <home>com.ibm.wssvt.tc.pli.ejb.WSMultiProtocolHome</home>
   <remote>com.ibm.wssvt.tc.pli.ejb.WSMultiProtocol</remote>
   <ejb-class>com.ibm.wssvt.tc.pli.ejb.WSMultiProtocolWebSvcsBean</ejb-class>
   <session-type>Stateless</session-type>
   <transaction-type>Container</transaction-type>
   <ejb-ref-id="EjbRef_1082407586720_2">
    <description></description>
    <ejb-ref-name>ejb/BeneficiarySession</ejb-ref-name>
```

```
    <ejb-ref-type>Session</ejb-ref-type>
    <home>com.ibm.wssvt.tc.pli.ejb.BeneficiarySessionHome</home>
    <remote>com.ibm.wssvt.tc.pli.ejb.BeneficiarySession</remote>
    <ejb-link>PolicySession20EJB.jar#BeneficiarySession</ejb-link>
   </ejb-ref>
   <ejb-ref-id="EjbRef_1082407586790_2">
    <description></description>
    <ejb-ref-name>ejb/PolicySession</ejb-ref-name>
    <ejb-ref-type>Session</ejb-ref-type>
    <home>com.ibm.wssvt.tc.pli.ejb.PolicySessionHome</home>
    <remote>com.ibm.wssvt.tc.pli.ejb.PolicySession</remote>
    <ejb-link>PolicySession20EJB.jar#PolicySession</ejb-link>
   </ejb-ref>


  <session-id="WSMultiProtocol_3">
   <ejb-name>WSMultiProtocol_3</ejb-name>
   <home>com.ibm.wssvt.tc.pli.ejb.WSMultiProtocolHome</home>
   <remote>com.ibm.wssvt.tc.pli.ejb.WSMultiProtocol</remote>
   <ejb-class>com.ibm.wssvt.tc.pli.ejb.WSMultiProtocolWebSvcsBean</ejb-class>
   <session-type>Stateless</session-type>
   <transaction-type>Container</transaction-type>
   <ejb-ref-id="EjbRef_1082407586790_3">
    <description></description>
    <ejb-ref-name>ejb/BeneficiarySession</ejb-ref-name>
    <ejb-ref-type>Session</ejb-ref-type>
    <home>com.ibm.wssvt.tc.pli.ejb.BeneficiarySessionHome</home>
    <remote>com.ibm.wssvt.tc.pli.ejb.BeneficiarySession</remote>
    <ejb-link>PolicySession20EJB.jar#BeneficiarySession</ejb-link>
   </ejb-ref>
   <ejb-ref-id="EjbRef_1082407586790_3">
    <description></description>
    <ejb-ref-name>ejb/PolicySession</ejb-ref-name>
    <ejb-ref-type>Session</ejb-ref-type>
    <home>com.ibm.wssvt.tc.pli.ejb.PolicySessionHome</home>
    <remote>com.ibm.wssvt.tc.pli.ejb.PolicySession</remote>
    <ejb-link>PolicySession20EJB.jar#PolicySession</ejb-link>
   </ejb-ref>
  </session>
```

Avoiding application errors after uninstalling an interim fix, a fix pack, or a refresh pack

If an application uses functions that are provided by a particular fix and you remove the fix, the application displays an error message. If you remove a fix, make sure that you retest your applications to check for errors. Redeploy any applications that display an error message because of the missing fix.

For example, suppose you install a fix pack on WebSphere Application Server and you create the stock quote Web service, StockQuote. The **WSDL2Java** command-line tool is used in a deployer role and generates a ServiceLocator class that extends the AgnosticService class.

If you uninstall the fix pack, the application is using a new Web services class (AgnosticService) that the version of WebSphere Application Server does not support. The application throws the following error:

```
java.lang.NoClassDefFoundError:
 Error while defining class:
 com.ibm.ws.wsfvt.test.stockquote.StockQuoteServiceLocator
 This error indicates that the class:
 com.ibm.webservices.multiprotocol.AgnosticService
 could not be located  while defining the class:
 com.ibm.ws.wsfvt.test.stockquote.StockQuoteServiceLocator
```

You need to redeploy the application on the WebSphere Application Server to emit code that does not use the WebSphere Application Server version that is not supported by the Web services class that you use.

Using a proxy server to access the Internet while running the WSDL2Java command causes your connection to time out

If you use an environment that requires a proxy server to access the Internet during the run of the **WSDL2Java** command, the **WSDL2Java** command might not find the Internet information because the proxy server has the potential to time out. For example, if the input WSDL file is located on the Internet instead of a local drive, and you need to retrieve it from the Internet, the **WSDL2Java** command fails to find the file because the proxy server times out.

You can work around this problem by editing the **WSDL2Java** command that is located in the `app_server_root/bin` directory. Set your proxy host and port value in one of following ways:

- Edit the `profile_root/bin/WSDL2Java` file and add the following line to the beginning of the file:

  ```
  export PROXY_INFO="-Dproxy.httpHost=yourProxyHost -Dproxy.httpPort=yourProxyPort"
  ```

  If you use this option, the **WSDL2Java** command, located in the `profile_root/bin` directory, must be invoked.
- Set the `PROXY_INFO` environment variable in the Qshell session prior to invoking the **WSDL2Java** command as follows:

  ```
  $ export PROXY_INFO ="-Dproxy.httpHost=yourProxyHost -Dproxy.httpPort=yourProxyPort"
  ```

  If you use this option, you need to run the command every time a new QSHELL session is started. If you do not want to start the QSHELL each time, you can add the command to the profile.

On Linux and AIX operating systems, edit the *install_root*`/bin/setupCmdLine.sh` file and add $*install_root*`/lib/urlprotocols.jar` to the end of the line that sets the *WAS_CLASSPATH* environment variable.

Troubleshooting Web services compiled bindings

This topic discusses troubleshooting compiled bindings of Web services that are developed and implemented based on the Web Services for Java 2 Platform, Enterprise Edition (J2EE) specification

Each section in this topic is a problem that you might experience with compiled bindings for Web services. A solution is provided to help you troubleshoot the problem.

Context root not recognized when mapping the default XML namespace to a Java package

When you map the default XML namespace to a Java package the context root is not recognized. If two namespaces are the same up to the first slash, they map to the same Java package. For example, the XML namespaces `http://www.ibm.com/foo` and `http://www.ibm.com/bar` both map to the `www.ibm.com` Java package . Use the -NStoPkg option of the **Java2WSDL** command to specify the package for the fully qualified namespace.

Java code to Web Service Description Language (WSDL) mapping cannot be reversed to the original Java code

If you find that a WSDL file that you created with the **Java2WSDL** command-line tool cannot be compiled when regenerated into Java code using the **WSDL2Java** command-line tool, it is because the Java API for XML-based remote procedure call (JAX-RPC) mapping from Java code to WSDL is not reversible back to the original Java code.

To troubleshoot this problem, try specifying the -introspect option to the **WSDL2Java** command. The -instrospect option indicates to the **WSDL2Java** command to look into existing Java classes and gather information useful in generating artifacts that match the original Java code.

Troubleshooting the run time for a Web services client

This topic discusses troubleshooting Web services clients.

Each section in this topic is a problem that you might experience during the run-time of a Web services client. A solution is provided to help you troubleshoot the problem.

Running your Web service client with an `ibm-jaxrpc-client.jar` file in a Solaris environment can cause an exception

If you use the **-jar** option, for example, `java -jar <java_application>.jar`, to specify a Java application in a Solaris environment, a class not found exception can occur. To avoid an exception, use the **-classpath** option instead of the **-jar** option, for example,

```
java -jar <your_client_application>.jar, <main_class_file>
```

.

The problem occurs because the Sun JDK classloading specification is more strict than the IBM JDK specifications.

You can make one of three changes to avoid an exception:

- Use the **-classpath** option instead of the **-jar** option, for example,

  ```
  java -jar <java_application>.jar, <main_class_file >
  ```

  .
- Use the **-Djava.ext.dirs** option with the **-jar** option, for example,

  ```
  export WAS_HOME=/opt/IBM/WebSphere/AppServer ${WAS_HOME}/java/jre/bin/java
  -Djava.ext.dirs=${WAS_HOME}/runtimes
  -jar <your_client_application>.jar, <your_client_application>.args
  ```

  .
- Modify Class Path in Manifest.MF to include the Java archive (JAR) files that you need, for example,

  ```
  Class Path: /opt/IBM/WebSphere/AppServer/runtimes/ibm-jaxrpc-client.jar
  ```

Resolving DNS causes performance problems when using HTTP to connect to a service endpoint interface that is not based on a private IP address

The DNS service is often not available when you use HTTP to connect to a service endpoint interface that is based on a private IP address. Therefore, performance is degraded during the DNS resolution.

This problem occurs when the outbound HTTP connector in the Web service engine attempts to resolve the host address name and times out.

You can modify the HOSTS file for the targeted IP address to avoid the DNS resolution.

Runtime migration error

If you installed a Web service application that was developed for a WebSphere Application Server version prior to Version 6, you might get the following exception:

```
WSWS3701E: Error: An exception was encountered. Use wsdeploy to deploy your application.
This may correct the problem. The exception is <exception data>.
```

This exception indicates that a problem occurred while running the application that was developed with tools supported by versions prior to Version 6. A solution to the problem is to uninstall the application, run the **wsdeploy** command and redeploy the application.

WebServicesFault exception displays during the application server run time for certain Web Services Description Language (WSDL) files

A `WebServicesFault` exception displays during the application server run time for WSDL files that define operations with `document style` and `literal use`, and use the SOAP header to transmit the input data.

If the WSDL files define the operation with `document style` and `literal use`, and this operation maps the input to the SOAP header, the Web services run time fails to find the correct operation for the target service and the `WebServicesFault` exception displays.

To solve the problem, change the WSDL files so that the operation does not have input that uses the SOAP header to transmit the data.

Increase the value of the ConnectionIOTimeOut parameter to avoid receiving an exception when hosting Web services on WebSphere Application Server

When hosting Web services on WebSphere Application Server, the following exception displays: `java.net.SocketTimeOutException: Read Timed Out`.

A slow network connection between the client and the Web service causes this problem. In such cases, the HTTP socket might time out before the Web service engine completely reads the SOAP request. In the majority of cases, a sudden increase in overall network activity causes this problem. The problem can also occur when the client is accessing the Web service from a slow network connection and when the SOAP request has a lot of data.

To solve the problem, increase the ConnectionIOTimeOut parameter for the Web container HTTP transport. The default value is 5 seconds. Increase the value to 30 seconds or greater. Type the following property name and value:

- **Name:** ConnectionIOTimeOut
- **Value:** 30

If the Web service is hosted in a clustered environment, set the property on each application server in the cluster. If your application server is listening on more than one port number, set the property on all ports. For more information about setting the value using the administrative console, refer to the HTTP transport custom properties chapter in the *Administering applications and their environment* PDF book.

Executing a Web services client application with session persistence turned on or in a clustered environment might cause a WebServicesFault error

When you execute a Web services client application with session persistence turned on or in a cluster environment, an error might display because the Web service client attempts to use a connection that has been closed by the HTTP server. The following is an example of the error:

```
[mm/dd/yy hh:mm:ss:ttt EST] 0000006e SystemErr     R WebServicesFault
 faultCode: {http://schemas.xmlsoap.org/soap/envelope/}Server.generalException
 faultString: java.io.IOException: Connection close: Read failed.Possible end of
stream encountered.
 faultActor: null
 faultDetail:
```

You can avoid this error by following one of two ways:

- Set the com.ibm.websphere.webservices.http.requestResendEnabled property to `true`, for example, com.ibm.websphere.webservices.http.requestResendEnabled=true. When this property is set to `true`, the Web services client is programmed to re-send the request if the request has failed. Monitor your client runtime if you change the property value, because the request might be sent twice.

  For example, if your client is a banking application, and you set the com.ibm.websphere.webservices.http.requestResendEnabled property to `true`, a transaction might be

posted twice to an account. For more detailed information on configuring this property, read the section, Configuring additional HTTP transport properties using the JVM custom property panel in the administrative console in the *Developing and deploying applications* PDF book.

- If you are using the IBM HTTP Server on an AIX or Linux operating systems, you can set the MaxSpareThreads property to the same value as the MaxClients property that is located in the `http.conf` file. For example, if the MaxClients=600, change the MaxSpareThreads to equal 600 (MaxSpareThreads=600).

  The advantage to choosing this way to avoid the error, is that the IBM HTTP Server does not shut down idle or near-idle connections. The disadvantage to this choice is that the IBM HTTP Server uses excess resources to keep extra threads available, even during periods of light activity. This choice can only be done on an AIX or Linux operating system.

Troubleshooting serialization and deserialization in Web services

This topic discusses problems that you can have when you perform serialization and deserialization in Web services.

Each section in this topic is a problem that you might experience while serializing and deserializing Web services. A solution is provided to help you troubleshoot the problem.

Time zone information in deserialized java.util.Calendar is not as expected

When the client and server are based on Java code and a `java.util.Calendar` instance is received, the time zone in the received `java.util.Calendar` instance might be different from that of the `java.util.Calendar` instance that was sent.

This difference occurs because the `java.util.Calendar` is encoded as an xsd:dateTime for transmission. An xsd:dateTime is required to encode the correct time (base time plus or minus a time zone offset), but is not required to preserve locale information, including the original time zone.

The fact that the time zone for the current locale is not preserved needs to be accounted for when comparing Calendar instances. The `java.util.Calendar` class equals method checks that the time zones are the same when determining equality. Because the time zone in a deserialized Calendar instance might not match the current locale, use the before and after comparison methods to test that two Calendars refer to the same date and time as shown in the following examples:

```
java.util.Calendar c1 = ...// Date and time in time zone 1
java.util.Calendar c2 = ...// Same date and equivalent time, but in time zone 2

// c1 and c2 are not equal because their time zones are different
if (c1.equals (c2)) System.out.println("c1 and c2 are equal");

// but c1 and c2 do compare as "not before and not after" since they represent
the same date and time
if (!c1.after(c2) & !c1.before(c2) {
   System.out.println("c1 and c2 are equivalent");
}
```

Mixing Web services client and server bindings causes errors

Web Services for Java 2 Platform, Enterprise Edition (J2EE) and the Java API for XML-based remote procedure call (JAX-RPC) specifications do not support *round-trip* mapping between Java code and a Web Services Description Language (WSDL) document for all Java types. For example, you cannot turn or serialize a Java Date into XML code and then turn it back or deserialize it into a Java Date. This action deserializes as Java Calendar.

If you have a Java implementation that you create a WSDL document from, and you generate client bindings from the WSDL document, the client classes can be different from the server classes even

though the client classes have the same package and class names. The Web service client classes must be kept separate from the Web service server classes. For example, do not place the Web service server bindings classes in a utility Java archive (JAR) file and then include a Web service client JAR file that references the same utility JAR file.

If you do not keep the Web services client and server classes separate, a variety of exceptions can occur, depending on the Java classes used. The following is a sample stack trace error that can occur:

```
com.ibm.ws.webservices.engine.PivotHandlerWrapper TRAS0014I: The following exception was
loggedjava.lang.NoSuchMethodError: com.ibm.wssvt.acme.websvcs.ExtWSPolicyData:
   method getStartDate()Ljava/util/Date;
not found
at com.ibm.wssvt.acme.websvcs.ExtWSPolicyData_Ser.addElements(ExtWSPolicyData_Ser.java: 210)
at com.ibm.wssvt.acme.websvcs.ExtWSPolicyData_Ser.serialize (ExtWSPolicyData_Swer.java:29)
at com.ibm.ws.webservices.engine.encoding.SerializationContextImpl.serializeActual
(SerializationContextImpl.java 719)
at com.ibm.ws.webservices.engine.encoding.SerializationContextImpl.serialize
(SerializationContextImpl.java: 463)
```

The problem is caused by using an interface as shown in the following example for the service endpoint interface in the service implementation:

```
package server:
public interface Test_SEI extends java.rmi.Remote {
 public java.util.Calendar getCalendar () throws java.rmi.RemoteException;
 public java.util.Date getDate() throws java.rmi.RemoteException;
}
```

When this interface is compiled and run through the **Java2WSDL** command-line tool, the WSDL document maps the methods as shown in the following example:

```
<wsdl:message name="getDateResponse">
 <wsdl:part name="getDateReturn" type="xsd:dateTime"/>
</wsdl:message>

<wsdl:message name="getCalendarResponse">
 <wsdl:part name="getCalendarReturn" type="xsd:dateTime"/>
</wsdl:message>
```

The JAX-RPC mapping implemented by the Java2WSDL tool has mapped both the java.util.Date and java.util.Calendar instances to the xsd:dateTime XML type . The next step is to use the generated WSDL file to create a client for the Web service. When you run the WSDL2Java tool on the generated WSDL, the generated classes include a different version of the server.Test_SEI interface, for example:

```
package server;
public interface Test_SEI extends java.rmi.Remote {
 public java.util.Calendar getCalendar() throws java.rmi.RemoteException;
 public java.util.Date getDate() throws java.rmi.RemoteException;
}
```

The client version of the service.Test_SEI interface is different from the server version in that both the getCalendar and getDate methods return java.util.Calendar. The serialization and deserialization code that the client expects is the client version of the service endpoint interface. If the server version inadvertently appears in the client CLASSPATH variable, at either compilation or run time, an error occurs.

In addition to theNoSuchMethod error, the IncompatibleClassChangeError and ClassCastException can occur. However, almost any run-time exception can occur. The best practice is to be diligent about separating client Web services bindings classes from server Web services bindings classes. Always place the client bindings classes and server bindings classes in separate modules. If theses binding classes are in the same application, place the bindings classes in utility JAR files that are not shared between modules.

Troubleshooting authentication and authorization for Web services security based on the Web Services for Java 2 Platform, Enterprise Edition (J2EE) specification

Web services are developed and implemented based on the Web Services for Java 2 platform, Enterprise Edition (J2EE) specification. The are several troubleshooting authentication and authorization considerations when you are securing Web services.

These Web services are developed and implemented based on the Web Services for Java 2 platform, Enterprise Edition (J2EE) specification. This topic discusses troubleshooting authentication and authorization issues to consider when you are securing Web services.

Authentication challenge or authorization failure is displayed

You might encounter an authentication challenge or an authorization failure if a thread switch occurs. For example, an application might create a new thread or a raw socket connection to a servlet might open. A thread switch is not recommended by the J2EE specification because the security context information is stored in thread local. When a thread switch occurs, the authenticated identity is not passed from thread local to the new thread. As a result, WebSphere Application Server considers the identity to be unauthenticated. If you must create a new thread, you must propagate the security context to the new thread. However, this process is not supported by WebSphere Application Server.

Web services security enabled application fails to start

When a Web services security-enabled application fails to start, you might receive an error message similar to the following:

```
[6/19/03 11:13:02:976 EDT] 421fdaa2 KeyStoreKeyLo E WSEC5156E: An exception
while retrieving the key from KeyStore object:
java.security.UnrecoverableKeyException: Given final block not properly padded
```

The cause of the problem is that the keypass value or password provided for a particular key in the key store is invalid. The key store values are specified in the `KeyLocators` elements of one of following binding files: `ws-security.xml`, `ibm-webservices-bnd.xmi` or `ibm-webservicesclient-bnd.xmi`. Verify that the keypass values for keys specified in the KeyLocators elements are correct.

Applications with Web services security enabled cannot interoperate between WebSphere Application Server Version 6.0.x and Version 5.0.2

Applications with Web services security enabled cannot interoperate between WebSphere Application Server Version 6.0.x and Version 5.0.2. When applications attempt to interoperate, a ″digest mismatch″ error is displayed. An error exists in the cannonicalization algorithm for XML digital signature, which is fixed in Version 5.1. For Web services security to interoperate between WebSphere Application Server Version 6 and Version 5.0.2, you must update your Version 5.0.2 application server. To update your Version 5.0.2 server, access the WebSphere Application Server Support Web site and download the latest fix pack for WebSphere Application Server, Version 5.0.2.

Tracing SOAP messages with tcpmon

This topic discusses tracing SOAP messages that request Web services by using the tcpmon tool.

You can use other trace tools to trace SOAP messages, similar to how you can trace Web services components. For further information, see the Tracing Web services chapter in the *Administering applications and their environment* PDF book.

It is not recommended that you use the tcpmon tool in a stressed environment. Tcpmon is only for monitoring SOAP messages in a lightweight environment.

You can trace SOAP messages exchanged between a client and the server by installing a monitor or sniffer application to capture the HTTP traffic between the two points. The WebSphere product provides a utility class, `com.ibm.ws.webservices.engine.utils.tcpmon`, to trace the SOAP messages. The `com.ibm.ws.webservices.engine.utils.tcpmon` class redirects messages from a port, records the messages, and forwards the messages to another port.

WebSphere Application Server typically listens on port 9080, or port 80 if you are using IBM HTTP Server. The tcpmon process can be configured to listen on a particular port, such as 9088, while redirecting messages to another port, such as 9080 or port 80. The client is redirected to use port 9088 to access Web services.

Redirecting an application client to a different port is done by changing the SOAP address in the client Web Services Description Language (WSDL) file to use port 9088 and then running the **wsdeploy** command-line tool on the client enterprise archive (EAR) file to regenerate the service implementation.

Trace SOAP messages in Web services by following the actions listed in the steps for this task section.

1. Set up a development and unmanaged client run-time environment for Web services. For detailed information about setting up a development and unmanaged run-time environment, consult the Developing Web services chapter of the *Developing and deploying applications* PDF book.
2. Run the **java -Djava.ext.dirs=%WAS_EXT_DIRS% com.ibm.ws.webservices.engine.utils.tcpmon** command. A window labeled TCPMonitor is displayed.
3. Configure the TCPMonitor to listen on port 9088 and forward messages to port 9080.
   a. In the Listen Port # field, enter `9088`.
   b. Click **Listener**.
   c. In the TargetHostname field, enter `localhost`.
   d. In the Target Port # field, enter `9080`.
   e. Click **Add**.
   f. Click the **Port 9088** tab that displays on the top of the page.

The messages exchanged between the client and server appear in the TCPMonitor Request and Response pane.

Save the message data and analyze it.

Web services security troubleshooting tips

To troubleshooting Web services security, review the configurations with assembly tools to match the client and server request and the response configurations.

Troubleshooting Web services security is best done by reviewing the configurations with assembly tools so that you can match up the client and server request and the response configurations. These configurations must match. A client request sender configuration must match a server request receiver configuration. For encryption to successfully occur, the public key of the receiver must be exported to the sender and this key must be configured properly in the encryption information. For authentication, you must specify the method used by the client in the login mapping of the server. The following includes a list of generic troubleshooting steps that you can perform. For more information about the assembly tools, read the assembly tools section of the *Developing and deploying applications* PDF book.

Steps for this task

1. Verify that the client security extensions and server security extensions match on each downstream call for the following senders and receivers:
   - Request sender and request receiver
   - Response sender and response receiver

2. Verify that when the **Add Created Time Stamp** option is enabled on the client-side that the server has the **Add Received Time Stamp** option configured. You must configure the security extensions with an assembly tool.
3. Verify that the client security bindings and the server security bindings are correctly configured. When the client authentication method is signature, make sure that the server has a login mapping. When the client uses the public key `cn=Bob,o=IBM,c=US` to encrypt the body, verify that this Subject is a personal certificate in the server key store so that it can decrypt the body with the private key. You can configure the security bindings using an assembly tool or the WebSphere Application Server administrative console.
4. Check the `SystemOut.log` file in the `${USER_INSTALL_ROOT}/logs/server1` directory (*server1* changes depending upon the server name) for messages that might provide information about the problem.
5. Enable trace for Web services security by using the following trace specification:
   ```
   com.ibm.xml.soapsec.*=all=enabled:com.ibm.ws.webservices.*=all=enabled:
   com.ibm.wsspi.wssecurity.*=all=enabled:com.ibm.ws.security.*=all=enabled: SASRas=all=enabled
   ```
   Type the previous three lines as one continuous line.

Errors when securing Web services

The following errors might occur when you secure Web services:

- "″CWWSI5061E: The SOAP Body is not signed″ error message displays"
- "″CWWSI5075E: No security token found that satisfies any one of the authentication methods″ error message displays" on page 201
- "″CWWSI5094E: No UsernameToken of trusted user was found or the login failed for the user while the TrustMode is BasicAuth″ error message displays" on page 202
- "″CWSCJ0053E: Authorization failed for /UNAUTHENTICATED...″ error message displays" on page 202
- "″WSWS3243I: Info: Mapping Exception to WebServicesFault.″ error message is displayed when you specify the value type local name and the URI for a token consumer or the token generator" on page 203
- ""Invalid URI: The format of the URI could not be determined″ error message might display when you use a Microsoft .NET client that accesses a Web service for WebSphere Application Server" on page 203
- "″WSEC6664E: Null is not allowed to PKIXBuilderParameters. The configuration of TrustAnchor and CertStoreList are not correct″ exception displays" on page 204
- "″WSE567: The incoming Username token must contain both a nonce and a creation time for the replay detection feature″ Microsoft .NET error displays" on page 204

″CWWSI5061E: The SOAP Body is not signed″ error message displays

Version 5.x application

Solution for 5.*x* applications only:

This error usually occurs whenever the SOAP security handler does not load properly, and does not sign the SOAP body not to be signed. The SOAP security handler is typically the first validation that occurs on the server-side, so a multitude of problems can cause this message to display. The error might be caused by invalid actor URI configurations. You can configure the actor Universal Resource Identifier (URI) at the following locations within the assembly tool:

From the Web services client editor within the assembly tool for client configurations:

- Click **Security Extensions > Client Service Configuration Details** and indicate the actor information in the **ActorURI** field.

- Click **Security Extensions > Request Sender Configuration section > Details** and indicate the actor information in the **Actor** field.

From the Web Services Editor within the assembly tool for server configurations:

- Click **Security Extensions > Server Service Configuration** section. Verify that the actor URI has the same actor string as the client-side.
- Click **Security Extensions > Response Sender Service Configuration Details > Details** and indicate the actor information in the **Actor** field.

The actor information on both the client and the server must refer to the same string. When the actor fields on the client and the server match, the request or response is acted upon instead of being forwarded downstream. The actor fields might be different when you have Web services acting as a gateway to other Web services. However, in all other cases, verify that the actor information matches on the client and server. When the Web services implementation is acting as a gateway and it does not have the same actor configured as the request passing through the gateway, this Web services implementation does not process the message from the client. Instead, it sends the request downstream. The downstream process that contains the correct actor string processes the request. The same situation occurs for the response. Therefore, it is important that you verify that the appropriate client and server actor fields are synchronized.

Additionally, the error can appear when you do not specify that the body is signed in the client configuration. To sign the body part of the message using the Web service client editor in the assembly tool, click **Security Extensions > Request Sender Configuration > Integrity** and select the message parts to sign.

″CWWSI5075E: No security token found that satisfies any one of the authentication methods″ error message displays

Version 5.x application

Solution for 5.*x* applications only:

Verify that the client and server login configuration information matches in the security extensions. Also, verify that the client has a valid login binding and that the server has a valid login mapping in the security bindings. You can check this information by looking at the following locations in the assembly tool:

From the Web services client editor within the assembly tool for client configurations:

- Click **Security Extensions > Request Sender Configuration > Login Configuration** verify the authentication method.
- Click **Port Binding > Security Request Sender Binding Configuration > Login Binding** verify the authentication method and other parameters.

From the Web Services Editor within the assembly tool for server configurations:

- Click **Security Extensions > Request Receiver Service Configuration Details > Login Configuration** and verify the authentication method.
- Click **Binding Configurations > Request Receiver Binding Configuration Details > Login Mapping** and verify the authentication method and other parameters.

Also, make sure that the actor URI specified on the client and server matches. You can configure the actor URI at the following locations within the assembly tool:

From the Web services client editor within the assembly tool for client configurations:

- Click **Security Extensions > Client Service Configuration Details** and indicate the actor information in the **ActorURI** field.

- Click **Security Extensions > Request Sender Configuration section > Details** and indicate the actor information in the **Actor** field.

From the Web services editor within the assembly tool for server configurations:

- Click **Security Extensions > Server Service Configuration** section. Make sure that the **Actor URI** field has the same actor string as the client side.
- Click **Security Extensions > Response Sender Service Configuration Details > Details** and indicate the actor information in the **Actor** field.

″CWWSI5094E: No UsernameToken of trusted user was found or the login failed for the user while the TrustMode is BasicAuth″ error message displays

Solution:

This situation occurs when you have IDAssertion configured in the login configuration as the authentication method. On the sending Web service, configure a trusted basic authentication entry in the login binding. Then, on the server side, verify that the trusted ID evaluator has a property set that contains the user name of this basic authentication entry. To configure the client for identity assertion, consult the following topics:

- Configuring the client for identity assertion: specifying the method
- Configuring the client for identity assertion: collecting the authentication method

To configure the server for identity assertion, consult these topics:

- Configuring the server to handle identity assertion authentication
- Configuring the server to validate identity assertion authentication information

Note: These topics about configuring clients and servers for identity assertion are located in the chapter, Securing Web services for Version 5.x applications using identity assertion authentication in the *Securing applications and their environment* PDF book.

″CWSCJ0053E: Authorization failed for /UNAUTHENTICATED...″ error message displays

The following authorization error occurs with UNAUTHENTICATED as the security name: `CWSCJ0053E: Authorization failed for /UNAUTHENTICATED while invoking (Home)com/ibm/wssvt/tc/pli/ejb/ Beneficiary findBeneficiaryBySsNo(java.lang.String):2 securityName: /UNAUTHENTICATED;accessID: null is not granted any of the required roles: AgentRole`

This situation occurs because a login configuration is not being configured or Web services Security is not configured from a client to a server. When the request arrives at the server and authentication information is not received, the UNAUTHENTICATED user is set on the thread. Authorization returns this error if there are any roles assigned to the resource except for the special ″Everyone″ role, which supports access by anyone.

If the client successfully authenticates to an Enterprise JavaBeans (EJB) file, but the EJB file calls a downstream EJB file that is not configured with Web services security or transport security, such as HTTP user ID and password, an error can occur for this downstream request. Using the assembly tool, verify that the enterprise archive (EAR) file for both client and server has the correct security extensions and security bindings. For more information, consult the following topics:

- Configuring the client security bindings using an assembly tool
- Configuring the security bindings on a server acting as a client using the administrative console
- Configuring the server security bindings using an assembly tool
- Configuring the server security bindings using the administrative console

Note: These topics about client and server security bindings are located in the chapter, Securing Web services for Version 5.x applications using XML digital signature in the *Securing applications and their environment* PDF book.

″WSWS3243I: Info: Mapping Exception to WebServicesFault.″ error message is displayed when you specify the value type local name and the URI for a token consumer or the token generator

The Value type URI is not required for the following predefined value type local names:

- Username token
- X509 certificate token
- X509 certificates in a PKIPath
- A list of X509 certificates and CRLs in a PKCS#7

If you specify one of the previous value type local names, do not enter a value for the Value type URI field.

″Invalid URI: The format of the URI could not be determined″ error message might display when you use a Microsoft .NET client that accesses a Web service for WebSphere Application Server

The following exception message might display when you use a Microsoft .NET client that accesses a Web service for WebSphere Application Server. Within WebSphere Application Server, Web services security is enabled and uses the ActorURI attribute.

```
Invalid URI: The format of the URI could not be determined.
```

Exception type:

```
System.UriFormatException
at System.Uri.Parse()
at System.Uri..ctor(String uriString, Boolean dontEscape)
at System.Uri..ctor(String uriString)
at Microsoft.Web.Services2.SoapInputFilter.CanProcessHeader(XmlElement header, SoapContext context)
at Microsoft.Web.Services2.Security.SecurityInputFilter.ProcessMessage(SoapEnvelope envelope)
at Microsoft.Web.Services2.Pipeline.ProcessInputMessage(SoapEnvelope envelope)
at Microsoft.Web.Services2.InputStream.GetRawContent()
at Microsoft.Web.Services2.InputStream.get_Length()
at System.Xml.XmlScanner..ctor(TextReader reader, XmlNameTable ntable)
at System.Xml.XmlTextReader..ctor(String url, TextReader input, XmlNameTable nt)
at System.Xml.XmlTextReader..ctor(TextReader input)
at System.Web.Services.Protocols.SoapHttpClientProtocol.ReadResponse(SoapClientMessage message,
WebResponse response, Stream responseStream, Boolean asyncCall)
```

Solution: This error occurs because Microsoft .NET Web Services Enhancements (WSE) Version 2.0 Service Pack 3 does not support a relative URI value for the ActorURI attribute. WSE Version 2.0 Service Pack 3 supports an absolute Uniform Resource Identifier (URI) for this attribute only. To work with a Microsoft .NET client, you must configure this attribute as an absolute URI. An example of an absolute URI is: abc://myWebService. An example of a relative URI is: myWebService.

To configure ActorURI attribute for use with WebSphere Application Server, use either the Rational Application Developer (RAD) or the Application Server Toolkit (AST) to complete the following steps:

1. Open the Web Service Editor, click the **Extensions** tab and expand **Server Service Configuration**.
2. Enter the full absolute URI in the Actor field.
3. Expand **Response Generator Service Configuration Details > Details**.
4. Enter the full absolute URI in the Actor field.

″WSEC6664E: Null is not allowed to PKIXBuilderParameters. The configuration of TrustAnchor and CertStoreList are not correct″ exception displays

Possible cause:

The certificate path setting is not configured properly.

Possible solution:

Configure the certificate path setting by completing the following steps:

1. In the administrative console, click **Security > Web services**.
2. Under the Default consumer binding heading, click **Signing information >** *configuration_name*.
3. Select either the **Trust any** or **Dedicated signing information** option.

   If you select the **Dedicated signing information** option, select both a trust anchor and a certificate store from the configurations that are provided in the drop-down lists.
4. Click **OK** and **Save** to the master configuration.

″WSE567: The incoming Username token must contain both a nonce and a creation time for the replay detection feature″ Microsoft .NET error displays

Scenario:

In this scenario, you have a Web services client for WebSphere Application Server and a Microsoft .NET Web service. The Microsoft .NET Web service has a ws-security constraint for a username token configured. The following exception is thrown from the Microsoft .NET server:

WSE567: The incoming username token must contain both a nonce and a creation time for the replay detection feature.

Cause:

By default, the Microsoft .NET Web service validates the nonce and the timestamp for the username token. However, it is optional for you to configure the nonce and timestamp properties for a Web service client that is using WebSphere Application Server.

Solution:

Complete the following steps to add the nonce and timestamp properties for a username token on a Web service client for WebSphere Application Server. These steps involve an assembly tool such as Rational Application Developer or the Application Server Toolkit. For more information about assembly tools, read the assembly tools section of the *Developing and deploying applications* PDF book.

1. Open the Web service client deployment descriptor and click the **WS-Binding** tab.
2. Expand the Security Request Generator Binding Configuration > Token Generator sections.
3. Click the name of the username token that you already created and click **Edit**.
4. In the Properties section of the Token Generator dialog, click **Add**.
5. Enter `com.ibm.wsspi.wssecurity.token.username.addNonce` in the Name field to provide the name of the nonce property.
6. Enter `true` in the Value field.
7. Click **Add**.
8. Enter `com.ibm.wsspi.wssecurity.token.username.addTimestamp` in the Name field to provide the name of the timestamp property.
9. In the Value field, enter `true`.

10. Click **OK** and save the client deployment descriptor.

For more information about configuring token generators, see Configuring token generators with an assembly tool topic in the *Securing applications and their environment* PDF book.

Tips for troubleshooting the Web Services Invocation Framework

A set of specific tips to help you troubleshoot problems you experience with the Web Services Invocation Framework (WSIF).

For information about resolving WebSphere-level problems, see Diagnosing and fixing problems.

To identify and resolve Web Services Invocation Framework (WSIF)-related problems, you can use the standard WebSphere Application Server trace and logging facilities. If you encounter a problem that you think might be related to WSIF, you can check for error messages in the WebSphere Application Server administrative console, and in the application server `stdout.log` file. You can also enable the application server debug trace to provide a detailed exception dump.

A list of the WSIF run-time system messages, with details of what each message means, is provided in Message reference for WSIF.

A list of the main known restrictions that apply when using WSIF is provided in WSIF - Known restrictions.

Here is a checklist of major WSIF activities, with advice on common problems associated with each activity:

Create service

Handcrafted Web Services Description Language (WSDL) can cause numerous problems. To help ensure that your WSDL is valid, use a tool such as WebSphere Development Studio for iSeries (WDS) to create your Web service. For more details about creating services with WSDL in an iSeries environment, read the WSIF and WSDL chapter of *Developing and deploying applications* PDF book.

Define transport mechanism

For the Java Message Service (JMS), check that you have set up the Java Naming and Directory Interface (JNDI) correctly, and created the necessary connection factories and queues.

For SOAP, make sure that the deployment descriptor file `dds.xml` is correct - preferably by creating it using WebSphere Development Studio for iSeries (WDS) or similar tooling.

Create client - Java code

Follow the correct format for creating a WSIF service, port, operation and message. For examples of correct code, refer to the Address Book Sample in the *Developing and deploying applications* PDF book.

Compile code (client and service)

Check that the build path against code is correct, and that it contains the correct levels of JAR files.

Create a valid EAR file for your service in preparation for deployment to a Web server.

Deploy service

When you install and deploy the service EAR file, check carefully any messages given when the service is deployed.

Server setup and start

Make sure that the WebSphere Application Server `server.policy` file (in the `/properties` directory) has the correct security settings. For more information about setting up security, see the Enabling security for WSIF topic in the *Securing applications and their environment* PDF book.

WSIF setup

Check that the `wsif.properties` file is correctly set up. For more information about this file, refer to the Maintaining the WSIF properties file topic in the *Administering applications and their environment* PDF book.

Run client

Either check that you have defined the class path correctly to include references to your client classes, WSIF JAR files and any other necessary JAR files, or (preferably) run your client using the WebSphere Application Server launchClient tool.

Either check that you have defined the class path correctly to include references to your client classes, WSIF JAR files and any other necessary JAR files, or (preferably) run your client using the WebSphere Application Server launch client tool. For more information about this tool, refer to the Running application clients chapter of the *Developing and deploying applications* PDF book.

Here is a set of tips to help you troubleshoot commonly-experienced problems:
- "No class definition" errors received when running client code.
- "Cannot find WSDL" error.
- Your Web service EAR file does not install correctly onto the application server.
- There is a permissions problem or security error.
- Using WSIF with multiple clients causes a SOAP parsing error.
- Using the same names for JMS messaging queues and queue connection factories that run on application servers on different machines can cause JNDI lookup errors.
- A JAX-RPC client running on WebSphere Application Server Version 5 uses SOAP over JMS to invoke a Web service running on a Version 5 application server. No user ID or password is required on the target MQ Series queue. After the application server is migrated to Version 6, and using Version 6 default messaging, client requests fail because basic authentication is now enabled.
- The current WSIF default SOAP provider (the IBM Web Service SOAP provider) does not fully interoperate with services that are running on the former (Apache SOAP) provider.

"No class definition" errors received when running client code.

This problem usually indicates an error in the class path setup. Check that the relevant JAR files are included.

"Cannot find WSDL" error.

Some likely causes are:
- The application server is not running.
- The server location and port number in the WSDL are not correct.
- The WSDL is badly formed (check the error messages in the application server `stdout.log` file).
- The application server has not been restarted since the service was installed.

You might also try the following checks:
- Can you load the WSDL into your Web browser from the location specified in the error message?
- Can you load the corresponding WSDL binding files into your Web browser?

Your Web service EAR file does not install correctly onto the application server.

It is likely that the EAR file is badly formed. Verify the installation by completing the following steps:
- For an EJB binding, run the WebSphere Application Server tool `\bin\dumpnamespace`. This tool lists the current contents of the JNDI directory.
- For a SOAP over HTTP binding, open the `http://pathToServer/WebServiceName/admin/list.jsp` page (if you have the SOAP administration pages installed). This page lists all currently installed Web services.
- For a SOAP over JMS binding, complete the following checks:
  - Check that the queue manager is running.
  - Check that the necessary queues are defined.

- Check the JNDI setup.
- Use the “display context” option in the `jmsadmin` tool to list the current JNDI definitions.
- Check that the Remote Procedure Call (RPC) router is running.

There is a permissions problem or security error.

Check that the WebSphere Application Server `server.policy` file (in the `/properties` directory) has the correct security settings. For more information about setting up security, see the Enabling security for WSIF topic in the *Securing applications and their environment* PDF book.

Using WSIF with multiple clients causes a SOAP parsing error.

Before you deploy a Web service to WebSphere Application Server, you must decide on the scope of the Web service. The deployment descriptor file `dds.xml` for the Web service includes the following line:

```
<isd:provider type="java" scope="Application" ......
```

You can set the `Scope` attribute to `Application` or `Session`. The default setting is `Application`, and this value is correct if each request to the Web service does not require objects to be maintained for longer than a single instance. If `Scope` is set to `Application` the objects are not available to another request during the execution of the single instance, and they are released on completion. If your Web service needs objects to be maintained for multiple requests, and to be unique within each request, you must set the scope to `Session`. If `Scope` is set to `Session`, the objects are not available to another request during the life of the session, and they are released on completion of the session. If scope is set to `Application` instead of `Session`, you might get the following SOAP error:

```
SOAPException: SOAP-ENV:ClientParsing error, response was:
FWK005 parse may not be called while parsing.;
nested exception is:

[SOAPException: faultCode=SOAP-ENV:Client; msg=Parsing error, response was:

FWK005 parse may not be called while parsing.;
        targetException=org.xml.sax.SAXException:
FWK005 parse may not be called while parsing.]
```

Using the same names for JMS messaging queues and queue connection factories that run on application servers on different machines can cause JNDI lookup errors.

You should not use the same names for messaging queues and queue connection factories that run on application servers on different machines, because WSIF always looks first for JMS destinations locally, and only uses the full JNDI reference if it cannot find the destination locally. For example, if you run a Web service on a remote machine, and have an application server running locally that uses the same names for the messaging queues and queue connection factories, then WSIF will find and use the local queues even if the remote JNDI destination is provided in full in the WSDL service definition.

A JAX-RPC client running on WebSphere Application Server Version 5 uses SOAP over JMS to invoke a Web service running on a Version 5 application server. No user ID or password is required on the target MQ Series queue. After the application server is migrated to Version 6, and using Version 6 default messaging, client requests fail because basic authentication is now enabled.

The problem appears as a log message:

```
SibMessage W [:] CWSIT0009W: A client request failed in the application server
with endpoint <endpoint_name> in bus <your_bus> with reason: CWSIT0016E:
The user ID null failed authentication in bus <your_bus>.
```

For the steps to take to resolve the problem, see the following service integration technologies troubleshooting tip: After you migrate an application server to WebSphere Application Server Version 6, existing Web services clients can no longer use SOAP over JMS to access services hosted on the migrated server. This tip can be found in the Tips for troubleshooting service integration bus security section.

The current WSIF default SOAP provider (the IBM Web Service SOAP provider) does not fully interoperable with services that are running on the former (Apache SOAP) provider.

This restriction is due to the fact that the IBM Web Service SOAP provider is designed to interoperate fully with a JAX-RPC compliant Web service, and Apache SOAP cannot provide such a service. To enable interoperation, modify either your Web service or the WSIF default SOAP provider. For information about how to modify your provider, read the chapter WSIF SOAP provider: working with legacy applications in the *Developing and deploying applications PDF* book.

Trace and logging for WSIF

The Web Services Invocation Framework (WSIF) offers trace points at the opening and closing of ports, the invocation of services, and the responses from services. WSIF also includes a `SimpleLog` utility that can run trace when you are using WSIF outside of WebSphere Application Server.

If you want to enable trace for the WSIF API within WebSphere Application Server, and have trace, stdout and stderr for the application server written to a well-known location, see Enabling tracing and logging.

To trace the WSIF API, you need to specify the following trace string:

```
wsif=all=enabled
```

To enable the WSIF `SimpleLog` utility, through which you can run trace when using WSIF outside of WebSphere Application Server, complete the following steps:

1. Create a file named `commons-logging.properties` with the following contents:

```
org.apache.commons.logging.LogFactory=org.apache.commons.logging.impl.LogFactoryImpl
org.apache.commons.logging.Log=org.apache.commons.logging.impl.SimpleLog
```

2. Create a file named `simplelog.properties` with the following contents:

```
org.apache.commons.logging.simplelog.defaultlog=trace
org.apache.commons.logging.simplelog.showShortLogname=true
org.apache.commons.logging.simplelog.showdatetime=true
```

3. Put both these files, and the `commons-logging.jar` file, on the class path.

The `SimpleLog` utility writes trace to the `System.err` file.

WSIF (Web Services Invocation Framework) messages

This topic contains a list of the WSIF run-time system messages, with details of what each message means.

WebSphere system messages are logged from a variety of sources, including application server components and applications. Messages logged by application server components and associated IBM products start with a unique message identifier that indicates the component or application that issued the message.

For more information about the message identifier format, see the topic Message reference.

WSIF0001E: An extension registry was not found for the element type "{0}"

Explanation: Parameters: {0} element type. No extension registry was found for the element type specified.

User Response: Add the appropriate extension registry to the port factory in your code.

WSIF0002E: A failure occurred in loading WSDL from "{0}"

Explanation: Parameters: {0} location of the WSDL file. The WSDL file could not be found at the location specified or did not parse correctly

User Response: Check that the location of the WSDL file is correct. Check that any network connections required are available. Check that the WSDL file contains valid WSDL.

WSIF0003W: An error occurred finding pluggable providers: {0}

Explanation: Parameters: {0} specific details about the error. There was a problem locating a WSIF pluggable provider using the J2SE 1.3 JAR file extensions to support service providers architecture. The WSIF trace file will contain the full exception details.

User Response: Verify that a META-INF/services/org.apache.wsif.spi.WSIFProvider file exists in a provider jar, that each class referenced in the META-INF file exists in the class path, and that each class implements org.apache.wsif.spi.WSIFProvider. The class in error will be ignored and WSIF will continue locating other pluggable providers.

WSIF0004E: WSDL contains an operation type "{0}" which is not supported for "{1}"

Explanation: Parameters: {0} name of the operation type specified. {1} name of the portType for the operation. An operation type which is not supported has been specified in the WSDL.

User Response: Remove any operations of the unsupported type from the WSDL. If the operation is required then make sure all messages have been correctly specified for the operation.

WSIF0005E: An error occurred when invoking the method "{1}" . ("{0}")

Explanation: Parameters: {0} name of communication type. For example EJB or Apache SOAP. {1} name of the method that failed. An error was encountered when invoking a method on the Web service using the communication shown in brackets.

User Response: Check that the method exists on the Web service and that the correct parts have been added to the operation as described in the WSDL. Network problems might be a cause if the method is remote and so check any required connections.

WSIF0006W: Multiple WSIFProvider found supporting the same namespace URI "{0}" . Found ("{1}")

Explanation: Parameters: {0} the namespace URI. {1} a list of the WSIFProvider found.. There are multiple org.apache.wsif.spi.WSIFProvider classes in the service provider path that support the same namespace URI.

User Response: A following WSIF0007I message will be issued notifying which WSIPFProvider will be used. Which WSIFProvider is chosen is based on settings in the wsif.properties file, or if not defined in the properties, the last WSIFProvider found will be used. See the wsif.properties file for more details on how to define which provider should be used to support a namespace URI.

WSIF0007I: Using WSIFProvider "{0}" for namespaceURI "{1}"

Explanation: Parameters: {0} the classname of the WSIFProvider being used. {1} the namespaceURI the provider will be used to support.. Either a previous WSIF0006W message has been issued or the SetDynamicWSIFProvider method has been used to override the provider used to support a namespaceURI.

User Response: None. See also WSIF0006W.

WSIF0008W: WSIFDefaultCorrelationService removing correlator due to timeout. ID:"{0}"

Explanation: Parameters: {0} the ID of the correlator being removed from the correlation service. A stored correlator is being removed from the correlation service due to its timeout expiring.

User Response: Determine why no response has been received for the asynchronous request within the timeout period. The wsif.asyncrequest.timeout property of the wsif.properties file defines the length of the timeout period.

WSIF0009I: Using correlation service - "{0}"

Explanation: Parameters: {0} the name of the correlation service being used. This identifies the name of the correlation service that will be used to process asynchronous requests.

User Response: None. If a correlation service other than the default WSIF supplied one is required, ensure that it is correctly registered in the JNDI java:comp/wsif/WSIFCorrelationService namespace.

WSIF0010E: Exception thrown while processing asynchronous response - "{0}"

Explanation: Parameters: {0} the error message string of the exception. While processing the response from an executeRequestResponseAsync call an exception was thrown.

User Response: Use the exception error message string to determine the cause of the error. The WSIF trace will have more details on the error including the exception stack trace.

WSIF0011I: Preferred port "{0}" was not available

Explanation: Parameters: {0} the user's preferred port. The preferred port set by the user on org.apache.wsif.WSIFService is not available

User Response: None unless this message appears for long periods of time in which case the user might want to pick a different port as their preferred port.

WSIF - Known restrictions

This topic lists the main known restrictions that apply when using WSIF.

Threading

WSIF is not thread-safe.

External Standards

WSIF supports:

- SOAP Version 1.1 (not 1.2 or later).
- WSDL Version 1.1 (not 1.2 or later).

WSIF does not provide WS-I compliance, and it does not support the Java API for XML-based Remote Procedure Calls (JAX-RPC) Version 1.1 (or later).

Full schema parsing

WSIF does not support full schema parsing. For example, WSDL references in complex types in the schema are not handled, and attributes are not handled.

XML Schema "redefine" elements are not handled and are ignored.

SOAP WSIF does not support:

- SOAP headers that are passed as `<parts>`.
- Unreferenced attachments in SOAP responses.
- Document Encoded style SOAP messages.

  Note: This is not primarily a WSIF restriction. Although you can specify Document Encoded style in WSDL, it is not generally considered to be a valid option and is not supported by the Web Services Interoperability Organization (WS-I).

SOAP provider interoperability

The current WSIF default SOAP provider (the IBM Web Service SOAP provider) does not fully interoperate with services that are running on the former (Apache SOAP) provider. This restriction is due to the fact that the IBM Web Service SOAP provider is designed to interoperate fully with a JAX-RPC compliant Web service, and Apache SOAP cannot provide such a service. For information on how to overcome this restriction, see WSIF SOAP provider: working with legacy applications.

WSIF's support for SOAP faults is restricted to SOAP faults originating from a Web service that runs using the IBM Web Service SOAP provider.

Note: This is not primarily a WSIF restriction. The current SOAP faults specification does not prescribe how to encode a SOAP fault so that it maps to a Java exception. Consequently, each Web service run-time environment currently decides on its own SOAP fault format. The IBM Web Service SOAP provider can understand its own response SOAP faults, but not the SOAP faults from another provider.

Type mappings

The current WSIF default SOAP provider (the IBM Web Service SOAP provider) conforms to the JAX-RPC type mapping rules that were finalized after the former (Apache SOAP) provider was created. The majority of types are mapped the same way by both providers. The exceptions are: `xsd:date`, `xsd:dateTime`, `xsd:hexBinary` and `xsd:QName`. Both client and service need to use the same mapping rules if any of these four types are used. Below is a table detailing the mapping rules for these four types:

XML Data Type	Apache SOAP Java Mapping	JAX-RPC Java Mapping
xsd:date	java.util.Date	Not supported
xsd:dateTime	Not supported	java.util.Calendar
xsd:hexBinary	Hexadecimal string	byte [ ]
xsd:QName	org.apache.soap.util.xml.QName	javax.xml.namespace.QName

Arrays and complex types

WSIF does not support general complex types, it only handles complex types that map to Java Beans. To use schema complex types, you must write your own custom serializers. The specific complex type and array support for WSIF outbound invocation of Web services is as follows:

- WSIF supports Java classes generated by WebSphere Studio Application Developer - Integration Edition (WSAD-IE) message generators (the normal case when WSDL files are downloaded from somewhere else). The WSAD-IE-based generation happens automatically when you use the BPEL editor, or the generation actions available on the Enterprise Services context menu, or the Business Integration toolbar.
- WSIF does not support Java beans generated by other tools, including the base WSAD tool.
- For WSAD-IE generated Java beans, attributes defined in the WSDL do not work. That is to say that these attributes, although they appear in the Java beans generated to represent the complex type, do not appear in the SOAP request created by WSIF.
- WSIF does not support arrays when they are a field of a Java bean. That is to say, WSIF only supports an array that is passed in as a named `<part>`. If an array is wrappered inside a Java bean, the array is not serialized in the same way.

Object Serialization

WSIF does not support serialization of objects across different releases.

Asynchronous invocation

WSIF supports synchronous invocation for all providers. For the JMS and the SOAP over JMS providers, WSIF also supports asynchronous invocation. You should call the `supportsAsync()` method before trying to execute an asynchronous operation.

The EJB provider

The target service of the WSIF EJB provider must be a remote-home interface, it cannot be an EJB local-home interface. In addition, the EJB stub classes must be available on the client class path.

Running outside WebSphere Application Server

WSIF is not supported for use outside WebSphere Application Server.

UDDI registry troubleshooting

When the UDDI registry is running, it might issue messages to report events or errors. You can use these messages as your first aid to problem determination. If you need more details about the cause of a problem, you can turn on tracing for UDDI. See the sub-topics below for information about messages, trace, and some common errors that you might encounter when setting up or using the UDDI registry.

If you cannot find the cause of a problem using these topics, use other knowledge bases as described in Searching knowledge bases.

Turning on UDDI registry trace

You can enable tracing of the UDDI registry at several levels of granularity. Please refer to Enabling tracing and logging for more information about using the administrative console to enable or disable trace; the component for the UDDI registry is `com.ibm.uddi`. For example, to enable all UDDI registry tracing, specify

```
com.ibm.uddi.*=all
```

Common causes of errors in the UDDI registry

Below are a few of the common causes of errors that might be found and their suggested solutions.

- The first start of the UDDI application may take some time to complete:
  - When you start the UDDI registry application for the first time with a new UDDI registry database, it must perform UDDI initialization, which occurs automatically for a default UDDI node, or when requested for a customized UDDI node. UDDI initialization populates the UDDI registry with pre-defined data and entities, and can therefore take some time to complete. This is expected behavior. Note that this only occurs on the first start, not subsequent starts, of the UDDI application.
  - If you use wsadmin to issue a command to start the UDDI application, then depending on your TCP timeout settings, this request might time out while waiting for UDDI to complete initialization. The UDDI initialization and the starting of the UDDI application will continue unaffected by this timeout, and will complete normally.
- There is a limitation concerning URL rewriting causing JavaScript syntax errors on several Web pages in the UDDI User Console. Because of this, cookies must be enabled in client browsers, the application server must have cookies enabled as the session tracking mechanism, and URL rewriting must be disabled.
- If you are attempting to use a remote DB2 database and are experiencing problems attaching to the remote system, one of the possible causes might be IP addressing. You should not have this problem if the remote system is using a static IP address, however if the remote system is using DHCP, the two systems must be aware of each others subnet mask.
- If you cannot see your UDDI node in the administrative console list of available nodes, check that the UDDI application is started on the relevant node/server.
- If you are unable to issue UDDI requests; for example, you start the UDDI user console and get errors when trying to publish or inquire, it is possible that:
  1. the database is not currently loaded or configured. Check the output from creating the database.
  2. the database is not correctly configured. Check the JDBC provider and datasource definitions are correct. This can be validated by using the Test Connection button on the administrative console.
  3. the UDDI node is not initialized. Check the UDDI node page on the administrative console. If the entry for the node in question does not show as activated, or deactivated, try to initialize the node having set any policies or properties first.
  4. the UDDI node is currently deactivated, while UDDI runtime settings are being updated. Check the UDDI node page on the administrative console. If the entry for the node in question shows as deactivated, then wait for it to change to activated, and then retry the request.

Reporting problems with the UDDI registry

If you report a problem with the UDDI registry component to IBM, supply the following information:

1. A detailed description of the problem. For example, if you were using a UDDI client, describe the client error seen and the type of client program, and provide details of the request issued. If you were using the UDDI user interface, describe the exact sequence you followed, and the result.
2. The build date and time of the version you are using. This can be obtained as follows:
   - In the *profile_root*/installedApps/*cell_name* subdirectory of the WebSphere installation location, you will find a subdirectory called UDDI_Registry.*node_name*.*server_name*.ear, where *node_name* is the

name of the node into which the UDDI registry application is installed, and *server_name* is the name of the server. Within that subdirectory, you will find a file called version.txt. Include the contents of this file as part of your information.
   - If the UDDI registry has been started with tracing enabled for the UDDI component, you should find a trace entry in the WebSphere trace log that includes the strings ″UDDIMessageLogger″ and ″UDDI Build :″ followed by the build date and time, and the build system. Also include this information.
3. Other environment information, such as:
   - Platform.
   - Database product.
   - Whether the database is local or remote, or for Cloudscape, whether it is embedded or networked.
   - Whether the server is in a standalone or a network deployment configuration.
   - Whether the UDDI node is default or customized.
   - The language used, for example, Chinese.
4. Any relevant log files and trace files.
   - If the problem occurred while setting up and installing the UDDI registry application using the setup script uddiDeploy.jacl, supply the log output from running the script. (If you did not redirect the output from the script file to a log file, rerun the script, this time redirecting the output as described in the sections Setting up a customized UDDI node, Setting up a default UDDI node with a default datasource or Setting up a default UDDI node.) The log file is written to the directory from which you ran the setup script. For detailed information on setting up UDDI nodes, see the chapter Setting up and deploying a new UDDI registry in the *Developing and deploying applications* PDF book.
   - If the problem occurred while removing the UDDI registry application using the remove script, uddiRemove.jacl, supply the log output from running the script. The log file is written to the directory from which you ran the remove script. For mode information about this task, see Removing a UDDI registry node in the *Administering applications and their environment* PDF book.
   - If the problem occurred while running the UDDI registry, enable UDDI tracing (if not already enabled) and supply the trace log from the logs directory of the application server on which the UDDI registry was running. See Turning on UDDI Trace for details on how to enable UDDI tracing. The Trace string that is most useful for initial analysis of problems by IBM is ″com.ibm.uddi.*=all″.
   - Also supply the WebSphere log files system.out and system.err.
   - Supply details of the version of IBM WebSphere Application Server you are running by running the command `versioninfo` on the application server node and directing the output to a log file.
5. If appropriate, any application code that you are using and the output produced by the application code.
6. UDDI utilizes First Failure Data Capture (FFDC) to capture data for unexpected UDDI errors.

   Run the WebSphere collector tool to collect the FFDC data and send the resulting jar to IBM. See Running the collector tool.

Chapter 12. Resolving in-doubt transactions

Use this task to resolve in-doubt transactions and the messages associated with them.

Transactions may become stuck in the in-doubt state indefinitely due to an exceptional circumstance such as the removal of a node causing messaging engines to be destroyed. When a transaction becomes in-doubt, it must be committed or rolled back so that normal processing by the affected messaging engine can continue.

You can use the administrative console to display the messages causing the problem (see Listing messages on a message point.) If there are messages involved in an in-doubt transaction, the identity of the transaction is shown in a panel associated with the message. You can then resolve the transaction in two ways:

1. Using the server's transaction management panels
2. Using methods on the messaging engine's MBean

You should first attempt to resolve the in-doubt transaction using the application server transaction management panels. You navigate to these panels in the administrative console by clicking **Servers** → **Application servers** → **[Content Pane]** *server-name* → **[Container Settings] Container Services** → **Transaction Service** → **Runtime** → **Imported prepared transactions - Review**. If the transaction identity appears in the resulting panel, you can commit or roll back the transaction. If the transaction identity does not appear in the panel, the transaction identity was not enlisted with the Transaction Service on the server. In this case only, you should use methods on the MBean to display a list of the identities of the in-doubt transactions managed directly by the messaging engine.

The following methods on the messaging engine's MBean can be used to get a list of transaction identities (xid) and to commit and roll back transactions:

- getPreparedTransactions()
- commitPreparedTransaction(String xid)
- rollbackPreparedTransaction(String xid)

To invoke the methods, you can use a wsadmin command, for example, you can use a command of the following form to obtain a list of the in-doubt transaction identities from a messaging engine's MBean:

```
wsadmin> $AdminControl invoke [$AdminControl queryNames type=SIBMessagingEngine,*] getPreparedTransactions
```

Note: You open a wsadmin command session from within Qshell. For more information, see the topic "Configure Qshell to run WebSphere Application Server scripts".

Alternatively, you can use a script such as the following to invoke the methods on the MBean:

```
set mbean [$AdminControl queryNames type=SIBMessagingEngine,*]
set input 0

while {$input >=0} {
  set xidList [$AdminControl invoke $mbean getPreparedTransactions]

  puts "----Prepared Transactions----"
  set index 0
  foreach xid $xidList {
    puts "  Index=$index XID=$xid"
    incr index
  }
  puts "------- End of list ---------"
  puts "Select index of XID to commit/rollback:"
  set input [gets stdin]

  if {$input < 0 } {
    puts "No index selected, existing."
```

```
  } else {
    set xid [lindex $xidList $input]
    puts "Enter c to commit or r to rollback XID $xid"
    set input [gets stdin]
    if {$input == "c"} {
      puts "Committing xid=$xid"
      $AdminControl invoke $mbean commitPreparedTransaction $xid
    }
    if {$input == "r"} {
      puts "Rolling back xid=$xid"
      $AdminControl invoke $mbean rollbackPreparedTransaction $xid
    }
  }
  puts ""
}
```

This script lists the transaction identities of the transactions together with an index. You can then select an index and commit or roll back the transaction corresponding to that index.

In summary, to identify and resolve in-doubt transactions:

1. Use the administrative console to find the transaction identity of messages that have in-doubt transactions.
2. **Optional:** If a transaction identity appears in the transaction management panel, commit or roll back the transactions as required.
3. **Optional:** If a transaction identity does not appear in the transaction management panel, use the methods on the messaging engine's MBean. For example, use a script to display a list of transaction identities for in-doubt transactions. For each transaction:
   a. Enter the index of the transaction identity of the transaction.
   b. **Optional:** Enter `c` to commit the transaction.
   c. **Optional:** Enter `r` to roll back the transaction.
4. To check that transactions are no longer in-doubt, restart the server and use the transaction management panel, or methods on the messaging engine's MBean to check.

Chapter 13. Learning about file stores

Use the subtopics to learn about various aspects of file stores, such as different types of files within it and high availability.

- "File stores"
- "File store high availability considerations" on page 218
- "File store configuration attributes"
- File store performance

File stores

File stores enable messaging engines to preserve operating information and to persist those objects that messaging engines need for recovery in the event of a failure, using a file system.

A file store is a type of message store which directly uses files in a file system via the operating system. File store mechanisms splits data storage into three levels: the log file, permanent store files and temporary store files. For more information on these files see "File store configuration attributes."

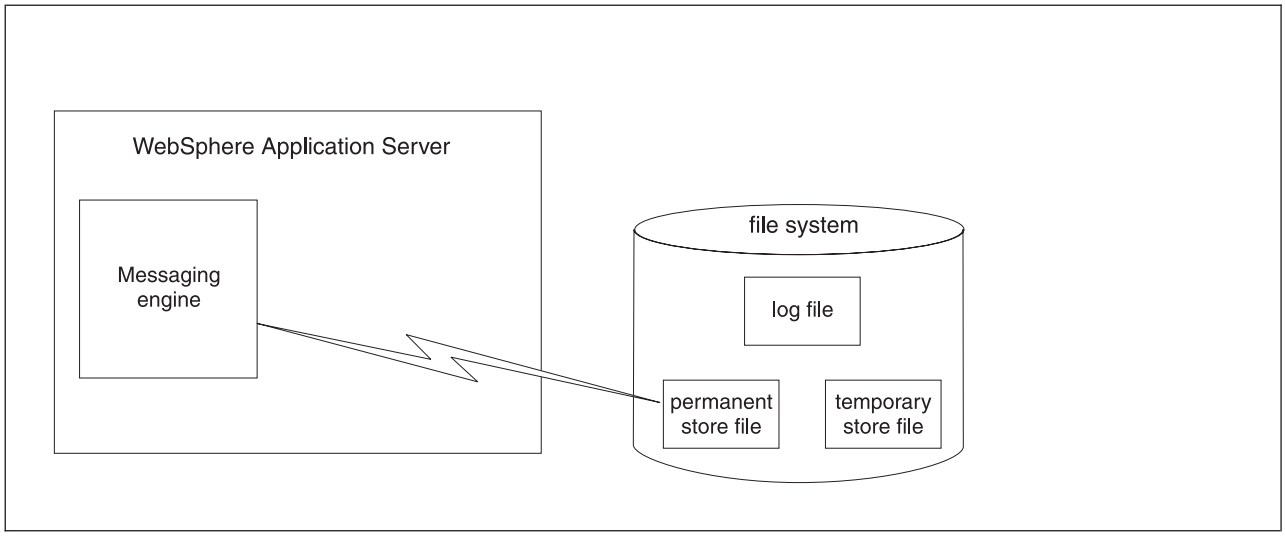


Figure 1. The relationship between a messaging engine and its file store.

File store configuration attributes

The log file, permanent store file and temporary store file make up a file store. You preserve appropriate amount of space within these three files so that operations and transactions behave predictably.

Data is first written to the log file sequentially, that is, new records are added to the end of the file. When the end of the log file is reached, old records at the beginning of the log file are overwritten by new records and this process repeats. Subsequently data is written to the permanent store file and temporary store file, although extremely short-lived data is only written to the log file.

The permanent store file and temporary store files have a minimum reserved sizes and a maximum size each. When created, the permanent and temporary store files consume their minimum reserved sizes, plus the size of the log. If the maximum size is larger, they grow up to the maximum size as required. The maximum size can be unlimited. It is recommended for production use that the minimum and maximum sizes are the same, as it prevents the store files from growing and shrinking at runtime so that the messaging engine does not gradually fill a file system. Another advantage is that if the file system does fill up while the messaging engine is operating, it is not affected.

The default configuration is intended to be sufficient to be used in typical messaging workloads without any administration. To improve the performance or availability of the log or the two store files, the administrator can modify the file store attributes to control where these files are placed. Similarly, the administrator can modify the attributes which control the sizes of the log and two store files to handle workloads with a large number of active transactions, large messages or a large volume of message data resident in the messaging engine

Note: This behavior cannot be guaranteed on a compressing file system, for example, NT file system with the **Compress this directory** option selected. You should avoid configuring file store to use a compressing file system for production use.

Table 2. File stores have the following attributes and values

Name	Description	Minimum and Default Values in mega bytes
Log size	Size of the log file, in mega bytes	• *Minimum*: 10 MB • *Default*: 100 MB
Minimum permanent store size	The minimum number of mega bytes reserved by the permanent store file. **Note:** The store files must always be at least as big as the log file.	• *Minimum*: 0 • *Default*: 200 MB
Maximum permanent store size	The maximum size in mega bytes of the permanent store file. **Note:** The store files must always be at least as big as the log file.	• *Minimum*: 50 MB • *Default*: 500 MB
Minimum temporary store size	The minimum number of mega bytes reserved by the temporary store file. **Note:** The store files must always be at least as big as the log file.	• *Minimum*: 0 • *Default*: 200 MB
Maximum temporary store size	The maximum size in mega bytes of the temporary store file. **Note:** The store files must always be at least as big as the log file.	• *Minimum*: 50 MB • *Default*: 500 MB
Unlimited permanent store size	Indicates whether the permanent store file is unlimited in size	• *Default*: false
Unlimited temporary store size	Indicates whether the temporary store file is unlimited in size	• *Default*: false
Log directory	Name of the directory that the log file is in	• *Default*: `${USER_INSTALL_ROOT}/filestores/com.ibm.ws.sib/<me_name>.<me_build>/log`
Permanent store directory	Name of the permanent store file's directory	• *Default*: `${USER_INSTALL_ROOT}/filestores/com.ibm.ws.sib/<me_name>.<me_build>/permanentStore`
Temporary store directory	Name of the temporary store file's directory	• *Default*: `${USER_INSTALL_ROOT}/filestores/com.ibm.ws.sib/<me_name>.<me_build>/temporaryStore`

File store high availability considerations

High availability refers to the capability of failing over messaging engines between servers. File stores can be used in highly available environments.

You can achieve high availability by choosing a file store as the message store of a messaging engine. In order to make a file store highly available, you should use hardware or software facilities to maximize the availability of the file store data, for example, SAN.

Note: It is important to ensure that the directories containing the log file and store files are universally accessible with the same directory name from all members of the cluster.

WebSphere Application Server v6.1 supports two styles of file system access to enable this:

- Cluster-managed file system

  This style of file system access uses high availability clustering and failover of shared disks to ensure that the file store's directories are accessible from the server that is currently running the messaging engine. The file store's directories are located on file systems in the shared disks, and high availability cluster scripts are used to mount the file systems on the node with the server that is running the messaging engine.
- Networked file system

  This style of file system access uses a network file system. The most popular protocols for accessing remote files are Common Internet File System (CIFS) and Network File System (NFS). It is recommended that you adopt Version 4 of NFS, which supports automated failover to ensure access locking. Access locking ensures the integrity of the log files, that is, only a single client process can access the log at a time.

Note: It is important to check that the file system configuration is correct, because it cannot be checked by the WebSphere configuration system or messaging engine. Errors only surface at runtime, so thorough failover testing is recommended.

Further information:
These considerations for enabling access to the file store's directories are similar to those for enabling access to the recovery log in a cluster. For more information see the following article: Transactional high availability and deployment considerations in WebSphere Application Server V6

Exclusive access to file store

A messaging engine has exclusive access to file store by design of file stores.

Each file store contains information which uniquely identifies the messaging engine which created it. A file store can only be used by the messaging engine which created it.

The messaging engine opens the file store's files in exclusive mode to prevent multiple instances of the same messaging engine from simultaneously using the file store, for example, by accidental activation on the messaging engine of multiple servers in a cluster. When a messaging engine stops, either in a controlled fashion or as a result of server failure, the file store's files are closed. Then another instance of the same messaging engine is able to open the file store.

Chapter 14. Messaging engine failover between v6 and v6.1

It is not permissible to failover a messaging engine using a file store onto a WebSphere Application Server v6 server. If you have a cluster as a bus member that consists of a mixture of v6 and v6.1 servers, you must modify the high availability policy to prevent this.

To prevent failover of a v6.1 messaging engine to a v6 server, the high availability policy for the messaging engine should be modified so that the cluster is effectively divided into sets of servers at the different versions and the messaging engine is restricted to the servers at v6.1.

Chapter 15. Tuning and problem solving for messaging engine data stores

Obtain an overview of improving the performance of messaging engine data stores and understanding problems that can occur with a data store.

For more information about tuning and problem solving for messaging engine data stores, see the following topics:
- Tuning the JDBC data source of a messaging engine
- Controlling the memory buffers used by a messaging engine
- Sharing connections to benefit from one-phase commit optimization
- "Diagnosing problems with data store exclusive access locks"
- "Diagnosing problems with your data store configuration" on page 224
- "Avoiding failover problems when you use DB2 v8.2 with HADR as your data store" on page 224

Diagnosing problems with data store exclusive access locks

Diagnose the causes of problems with data store exclusive access locks and examine possible causes to the problems.

Each messaging engine establishes an exclusive lock on its data store. While the messaging engine is running, it maintains that lock to ensure the integrity of the data within the data store.

Compare your symptoms with those listed in the following table and examine the possible solutions:

Symptom	Cause	Solution
The messaging engine cannot start. The messaging engine writes error message CWSIS1519	The messaging engine cannot connect to the database that you specified in the data source that you configured to enable the messaging engine to access its data store.	• Check that connectivity to the database is possible using the defined data source.
The messaging engine fails to start and writes error messages CWSIS1535 and CWSIS1519	The identifiers in the SIBOWNER table do not match those of the messaging engine.	• Check that the data source that you configured for the messaging engine refers to the correct database. • If the **MEUUID** identifiers do not match, check that a previous messaging engine did not use the same tables. If the tables already exist, DROP the tables and CREATE them again for the new messaging engine. • If the **INCUUID** identifiers do not match, another instance of the same messaging engine is running and has acquired the lock. Check for other running instances of the messaging engine.
The messaging engine starts but then stops and writes error message CWSIS1519.	The messaging engine has lost its lock on the data store.	• Check that you have connectivity to the database through the data source that you specified. The messaging engine might have lost network connectivity and be unable to maintain a connection with the database. • If you can connect to the database, another instance of the messaging engine might have started and obtained a lock on the data store. Check for other running instances of the messaging engine.

Diagnosing problems with your data store configuration

Find out how to diagnose problems that are caused by your data store configuration and possible solutions to these problems.

The following problems depend on the database that you use with your data store configuration and the level of that database:

- Examine this section if your messaging engine uses an Oracle 9i database for its data store and your messaging engine fails to start. If the messaging engine fails with the following message, where `XXXXXXXX` is the schema for the table, ensure that your Oracle installation is at 9.2.0.4, or higher:

  ```
  CWSIS1530E: The data type, 1,111, was found instead of the expected type, 2,004, for column,
  LONG_DATA, in table, XXXXXXXX.SIB000.
  ```

- Examine this section if your messaging engine uses a Sybase database for its data store. When you create your Sybase server:
  - Ensure that you create the database server with a page size of at least 4k.
  - Ensure that you set the **lock scheme** property on your server to the value *datarows*. This avoids the possibility of a deadlock on the data store tables.
- Examine this section if your messaging engine uses an Informix database for its data store and the messaging engine is unable to access its data store. When you configure your messaging engine to use an Informix database, ensure that you specify the schema name in lower case. For a full description of the configuring procedure, refer to Modify data store configurations.

Avoiding failover problems when you use DB2 v8.2 with HADR as your data store

Use this task to avoid problems that can occur when a messaging engine that is configured to use DB2 v8.2 with the High Availability Data Recovery (HADR) feature for its data store terminates if the DB2 database fails over.

If you use the High Availability Data Recovery (HADR) feature of DB2, note the following restrictions:

- The messaging engine default messaging provider supports only the synchronous and near-synchronoous synchronization modes of HADR. The default messaging provider does not support asynchronous HADR configurations.
- The TAKEOVER BY FORCE command is permitted only when the standby database is in peer state, or in a non-peer state (such as disconnected state) having changed from peer state.

One-phase commit optimization tuning

If you have configured your messaging engine to use a data store, you can achieve better performance by configuring both the messaging engine and container-managed persistent (CMP) beans.

You need to configure both the CMP and the messaging engine's resource authorization so that they share the same data source.

1. Open the administrative console.
2. Click on **Enterprise Applications** > *servername* > **Map data sources for all 2.x CMP beans**.
3. On the content pane, select the check boxes next to all the CMPs.
4. Select *Per application* in the **Resource authorization** selection list.
5. You can modify the messaging engine's resource authorization to *Per application* by modify the property file `sib.properties` by adding the custom property `sib.msgstore.jdbcResAuthForConnections=Application`.

Chapter 16. Listing messages on a message point

Use this task to list the messages that exist on a message point for a selected bus destination or messaging engine.

To display a list of messages on a message point, use the administrative console to complete the following steps:

1. In the navigation pane, click **Service integration** → **Buses**.
2. In the content pane, click the name of the service integration bus.
3. **Optional:** To list the message points for a bus destination, complete the following steps:
   a. In the content pane, under **Destination resources**, click **Destinations**
   b. Click the destination name.
4. **Optional:** To list the message points for a messaging engine, complete the following steps:
   a. In the content pane, under **Topology**, click **Messaging engines**
   b. Click the messaging engine name.
5. Under Additional Properties, click **Message points** This displays a list of message points in the content pane.
6. Click the message point name. This displays the properties of the destination localization in the content pane.
7. Click the Runtime tab.
8. Under Additional Properties, click **Messages**

A list of messages on the selected message point is displayed in the content pane.

You can select one or more messages to act on; for example, to display the message content, delete messages.

Chapter 17. Deleting messages on a message point

Use this task to delete one or messages that exist on a message point for a selected bus destination or messaging engine.

You should not normally need to delete messages on a message point. This task is intended as part of a troubleshooting procedure.

To delete one or messages on a message point, use the administrative console to complete the following steps:

1. List the messages on the message point.
2. In the content pane, click the check box next to each message you want to delete. Alternatively, you can select all messages in the list by clicking the Select all items button 
3. Click **Delete**

The selected messages are removed from the list.

Chapter 18. Resolving locked messages on a message point

Use this task to identify and resolve messages that are locked on a message point.

To resolve locked messages on a message point, use the administrative console to complete the following steps:

1. List the messages on the message point. The State property indicates whether or note the message is locked.
2. To list extra information about a message, click its name in the list. This displays the Messages settings panel, which shows a range of properties about the message; for example

   Time stamp
   The time stamp of the message.

   Message wait time
   The time the message has been waiting to be consumed.

If appropriate, after resolving the reason for messages being locked, you can delete messages on the Messages collection page.

Chapter 19. Troubleshooting service integration technologies

Use this overview task to help resolve a problem that you think is related to service integration technologies.

For general information about fixing problems, see Diagnosing and fixing problems.

To help you identify and resolve problems, use the WebSphere Application Server tracing and logging facilities. For more information about using tracing and logging, see Adding logging and tracing to your application.

If you encounter a problem that you think might be related to service integration technologies, complete the following stages.

1. Check the "Tips for troubleshooting" related links for specific problems related to service integration technologies.
2. Use "Troubleshooting service integration message problems" on page 254 to investigate problems with messages, such as messages not arriving or being consumed. If the tips and the investigations do not help you fix the problem, complete the following general stages.
3. Check the Release Notes for known problems and their resolution. The Release Notes are available from the WebSphere Application Server library web site.
4. For current information from IBM Support on known problems and their resolution, see the WebSphere Application Server support page.
5. Search the WebSphere Technotes database.

   You can use the following queries to find articles for service integration technologies:

   - All technotes: http://www.ibm.com/support/search.wss?rs=180tc=SSEQTP&tc1=SSCBRCS
   - MustGather technotes, used to debug problems: http://www.ibm.com/support/search.wss?rs=180tc=SSEQTP&tc1=SSCBRCS&q=MustGather
6. Check for error messages.

   Check in the application server's SystemOut log at *profile_root*\logs*server_name*\SystemOut.log for error messages, where *profile_root* is the directory in which profile-specific information is stored.

   The following message prefixes relate to specific aspects of service integration technologies:

Message prefix	Aspect of service integration technologies
CWSIA	API
CWSIB	Message formatting and parsing core
CWSIC	Communications
CWSID	Administration and system management
CWSIE	Message formatting and parsing SPI
CWSIF	Message formatting and parsing
CWSIG	Example
CWSIH	Match space
CWSII	Security
CWSIJ	Communications formats and protocol
CWSIK	Common messages
CWSIL	Publish and subscribe bridge
CWSIM	Mediations
CWSIN	Mediation services

Message prefix	Aspect of service integration technologies
CWSIO	Administration migration
CWSIP	Message processor
CWSIQ	MQ formats and protocol
CWSIR	Core programming interface
CWSIS	Message store
CWSIT	Topology routing and management
CWSIU	Utilities
CWSIV	Resource adapter
CWSIX	Core beans
CWSIY	Mediation handler framework
CWSIZ	Mediation framework
CWSJA	Administration commands
CWSJB	Inter-bus link
CWSJC	Core selector
CWSJD	Administration security
CWSJO	Service Data Objects configuration
CWSJQ	Message formatting and parsing MQ interoperability
CWSJR	Resource adapter (JMS)
CWSJU	Message tracing
CWSJW	WLM classifier for z/OS

The Troubleshooter reference: Messages contains information about the messages, indexed by message prefix. For each message there is an explanation of the problem, and details of any action that you can take to resolve the problem.

7. Check for more information and error messages that might provide a clue to a related problem. For example, if you see WebSphere MQ error messages or reason codes in WebSphere Application Server messages and logs, refer to the WebSphere MQ Messages document at http://publibfi.boulder.ibm.com/epubs/pdf/amqzao05.pdf.
8. If the information obtained in the preceding stages is still inconclusive you should explore further, for example by enabling the application server debug trace to provide a detailed exception dump.

Tips for troubleshooting service integration messaging

This topic provides a set of specific tips to help you troubleshoot problems with service integration messaging.

- “Enabling an application to be started before a required messaging engine has started” on page 237
- “Messages appear following activation of a message-driven bean during server startup” on page 237

Messaging engine cannot startup because of a known error in the Informix JDBC Driver 3.00JC1

When attempting to use the Informix JDBC driver 3.00JC1 to store data, the messaging engine cannot start up, and the following error message might appear in the WebSphere Application Server SystemOut.log file:

```
00000022 SibMessage E [RetireBus:retire_web.000- RetireBus] CWSIS0002E:
The messaging engine encountered an exception while starting.
Exception: com.ibm.ws.sib.msgstore.PersistenceException: CWSIS1501E:
The data source has produced an unexpected exception: java.sql.BatchUpdateException: Unique constraint
(informix.u114_62) violated.
00000022 SibMessage E [RetireBus:retire_web.000- RetireBus] CWSID0035E:
Messaging engine retire_web.000-RetireBus cannot be started;
detected error reported during com.ibm.ws.sib.msgstore.impl.MessageStoreImpl start()
00000022 SibMessage E [RetireBus:retire_web.000- RetireBus] CWSID0027I:
Messaging engine retire_web.000-RetireBus cannot be restarted because a serious error has been reported.T]
00000022 SibMessage I [RetireBus:retire_web.000- RetireBus] CWSID0016I:
Messaging engine retire_web.000-RetireBus is in state Stopped.
```

There is a known defect (PTS 172471) in the Informix JDBC Driver 3.00JC1. To avoid this error, upgrade the Informix JDBC Driver to 3.00JC2.

Problem determination for a data store

You can perform a dump, in reduced form, of the data in the data store for a messaging engine. The output is intended for use by IBM Service personnel.

If there is a problem with the data in the data store, it can be hard to diagnose from the trace output. However, you can create a dump, in XML format, of the data in the data store. This makes diagnosis easier because it is a human readable representation that can be transformed to other formats as required.

The dump is created as an XML file in the $WAS_HOME/logs/server1 directory. The file is named according to the format: *messaging_engine_nameUUIDtimestamp*.xml

The format of the file is illustrated in the following example:

```
<MessageStore>
    <itemStreams>
        <ItemStreamLink id="0" state="Available">
            <class>com.ibm.ws.sib.msgstore.ItemStream</class>
            <priority>5</priority>
            <canExpireSilently></canExpireSilently>
            <storageStrategy>STORE_NEVER</storageStrategy>
            <expiryTime>0</expiryTime>
            <sequence>0</sequence>
            <tranID>null</tranID>
            <tickValue>0</tickValue>
            <items>
                <ItemLink id="2" state="Available" refCount="3" refCountDecreasing="false">
                    <class>com.ibm.ws.sib.msgstore.Item</class>
                    <priority>5</priority>
                    <canExpireSilently></canExpireSilently>
                    <storageStrategy>STORE_NEVER</storageStrategy>
                    <expiryTime>0</expiryTime>
                    <sequence>1</sequence>
                    <tranID>null</tranID>
                    <tickValue>0</tickValue>
                </ItemLink></items></ItemStreamLink></itemStreams></MessageStore>
```

Possible causes of the XAResourceNotAvailableException exception and how to take appropriate action

When the deleteNode command is used for a node that hosts messaging engines, those messaging engines are deleted. When new messaging engines are re-created following the addNode command, they have different identifiers and so during transaction recovery it is not possible to connect to the old messaging engines. A message identifying the XAResourceNotAvailableException exception is generated in the SystemOut.log file for each server that hosts a messaging engine.

To solve this problem, you must follow the procedure described in Chapter 12, “Resolving in-doubt transactions,” on page 215.

Problems when you re-create a service integration bus

If you delete a service integration bus, and later create a new bus with the same name, the messaging engine fails to start and messages like the following are generated in SystemOut.log:

```
[8/11/04 21:55:01:439 CDT] 0000000f SibMessage    I
[LateBus:xyzsun15.server1-LateBus] isAlive: MessagingEngine suffered common mode error. Correct error (see
logs) and restart server.
[8/11/04 21:55:01:468 CDT] 0000000f SibMessage    I
[LateBus:xyzsun15.server1-LateBus] isAlive: MessagingEngine will be stopped because of common mode error.
No failover will occur.
[8/11/04 21:55:01:493 CDT] 0000000f SibMessage    I
[LateBus:xyzsun15.server1-LateBus] Messaging Engine
xyzsun15.server1-LateBus not in state from which stop is valid: Starting
[8/11/04 21:55:01:513 CDT] 0000000f SibMessage    I
[LateBus:xyzsun15.server1-LateBus] isAlive: MessagingEngine stopped because of common mode error. Correct
error (see logs) and restart server.
[8/11/04 21:57:01:431 CDT] 0000000e SibMessage    I
[LateBus:xyzsun15.server1-LateBus] isAlive: MessagingEngine suffered common mode error. Correct error (see
logs) and restart server.
```

The reason is that the database directory for the messaging engine still exists after deletion of the bus and must be manually removed. To delete the Cloudscape Version 10.1.x (Derby) database for a non-existent messaging engine, you must delete the database directory that is located in *profile_root*/databases/com.ibm.ws.sib, where *profile_root* is the directory in which profile-specific information is stored.

You must stop WebSphere Application Server before you can delete the database files.

For other databases, you can either delete all of the rows from the data store tables, or you can drop all of the tables. The names of the data store tables all begin with *SIB*, and are in the schema that you configured for the data store.

For more information , see Data store life cycle.

Problems when re-creating bus members

If you previously created a bus and added a bus member using the DEFAULT data source, when you attempt to recreate this bus member and you therefore delete the bus member and then try to add it again, the following exception appears:

```
ADMG0037E: A new instance of the DataSource object  cannot be created because the jndiName attribute
 of an existing DataSource object has the same value as jdbc/com.ibm.ws.sib/<NODENAME>.<SERVERNAME>-
<BUSNAME>
```

Note: The bus member is recreated but the messaging engine fails to start.

To resolve the problem, delete the data source manually. To do this, use the administrative console to complete the following steps:

1. Delete the bus member that is causing the exception.
2. Select **Resources** > **JDBC Providers**.
3. Change the scope to the scope of the data source that you want to delete. For example, on a single server install, you would choose Server, but this may vary with other topologies.
4. In the list of data sources, click Cloudscape Version 10.1.x (Derby) JDBC Provider. Note: Select the non XA option.
5. On the Configuration tab, under Additional Properties, click Data sources.
6. Select the check box next to the data source specified in the error message.
7. Click Delete.
8. Save your changes to the master configuration.

You should now be able to recreate the bus member without difficulty.

Problems communicating with foreign buses

To enable communication between buses, a foreign bus and a service bus integration link must be created. On the first bus, the name of the foreign bus must match the name of the second bus that becomes a foreign bus, and the name of the foreign bus for this second bus must match the name of the first bus. The service integration bus link must be have the same name on both buses.

You may see the following type of error if your configuration is not correct, for example because the service integration bus links do not match:

```
SibMessage    E   [TechBus:TechCluster.000-TechBus] CWSIT0057E: The inter-bus
connection BookstoreBus failed in the remote messaging engine on host
aixp401.rchland.ibm.com with reason: CWSIT0067E: Inter-bus connection BookstoreBus
in bus BookstoreBus is not available.
```

Problems when attempting to communicate with a renamed foreign bus

The administrative console panel used for configuring the properties of a service integration bus link, also allows you to change the foreign bus name that the link is pointing to. However, the foreign bus name must not be altered once it has been configured. If it is, any messaging engines that already hold state information about the link will not be able to use the link unless the foreign bus name is reset to its previous value.

Possible causes of a JMSException with a wrapped SILimitExceeded exception

When the number of messages held by a destination reaches its limiting threshold, any attempt to send a message to that destination fails with a JMSException with a wrapped SILimitExceeded exception. The destination continues to fail with this exception until the number of messages held by the destination reduces below the limiting threshold.

To obtain an accurate count of the number of available message, you can monitor the Available Message Count PMI statistic for queue and topicspace destinations. If the number of available messages increases, take action to balance the system. Consider stopping producers from sending new messages until the destination consumes the available messages.

Examine the following list for possible causes and solutions for this problem:

- The high threshold for the destination is too low for the projected number of messages. The destination does not process some messages. The default value for the high threshold is 50000.

  Solution: Increase the high threshold for the destination.
- Applications are producing more messages than the destination can process.

The ideal balance is for the number of messages produced and the number of messages consumed to be equal over a period of time. If your system is unbalanced and the producing application sends more messages than the destination can consume, the producing application eventually throws a JMSException.

Solution: Aim for a balance between the number of messages produced and the number of messages consumed.

Tip: The default setting for the Object Request Broker (ORB) thread pool is 100 threads. For some applications, this might allow 100 applications to send messages to the same destination. Consider tuning the ORB thread pool to have a maximum of 10 threads. This lower setting reduces the number of producers that can send messages, which might increase the overall message throughput.

- Applications are processing messages from the destination too slowly.

  Solution: It might be necessary to increase the number of messages consumed by the client applications. A destination processes more message when multiple consumers read from that destination.

  Consider cloning your application across multiple servers in a non-clustered environment. By default, applications are cloned in a clustered server environment. To enable subscribers in a non-clustered environment, set the **cloned** flag in the TopicConnectionFactory JNDI setting for DurableSubscriptions.

  Restriction: This solution is not suitable for applications that require total message ordering.

- Messages have a quality of service attribute that is *better than* best effort nonpersistent.

  Solution: Use messages for which the quality of service attribute is best effort nonpersistent. If there are too many messages in the system, the destination discardsbest effort nonpersistent messages.

  Restriction: This solution is not suitable for applications that must receive all messages.

Corruption problems on system restarts

It is possible, although rare, for a messaging engine, destination or link to be corrupted after a restart of the system. If this occurs you will see a message indicating the problem. If the problem lies with the messaging engine, the messaging engine will not start. If a destination or link is corrupted, the relevant messaging engine will start, but the destination or link will not be usable on that messaging engine.

If you do not know the cause of the problem, contact your IBM service representative to establish the cause before attempting to resolve the situation.

If you know the cause of the problem, for example, you are aware of an issue with your database, resolve it by completing the following steps:

1. Ensure that the configuration files are synchronized across your system by clicking **System administration** → **Nodes** → **Full resynchronize**. This operation can take several minutes to run.
2. If the problem still exists, perform one of the following tasks:
   - Delete the corrupted object and recreate it. Messages produced or received before the corruption occurred will be lost.
   - Restore your system from a backup, see Restoring a data store backup and recovering a messaging engine afterwards. Messages produced or received since the backup was taken will be lost.

Retrieving the status of messaging engines in the administrative console

To be able to retrieve the status of messaging engines, you must be logged into the administrative console with at least monitor authority. If you do not have this authority, the messaging engine status is displayed as ″Unavailable″, even if the messaging engine has started.

If you are not logged in with the authority needed to retrieve the status of messaging engines, an error message like the following is logged in the server's systemOut log file:

```
[4/20/05 10:49:57:083 CDT] 0000004b RoleBasedAuth A   SECJ0305I: The role-based  authorization check
failed for admin-authz operation SIBMessagingEngine:stateExtended. The user UNAUTHENTICATED (unique ID:
unauthenticated) was not granted any of the following required roles: administrator, operator,
configurator, monitor.
```

Where the user ID actually shown in the message is the user ID that you used to log in to the administrative console.

Enabling an application to be started before a required messaging engine has started

If an application depends on a messaging engine being available, then the messaging engine must be started before the application can be run. If you want application server to start an application automatically, you should develop your applications to test that any required messaging engine has been started and, if needed, wait for the messaging engine. If this is technique used in a startup bean, then the startup bean method should perform the test and wait work in a separate thread (using the standard WorkManager methods), so that the application server startup is not delayed.

For an example of code to test and wait for a messaging engine, see Enabling an application to be started before a messaging engine.

Messages appear following activation of a message-driven bean during server startup

If a message-driven bean is configured to connect to a given bus, but there are no suitable active messaging engines within the local server, the following message appears during server startup:

```
CWSIV0759W: During activation of a message-driven bean, no suitable active messaging engines were found in
the local server on the bus {0}.
```

Message consumption will begin when a suitable local messaging engine becomes active. Ensure that a suitable messaging engine is in the process of starting, in which case a message is displayed when a successful connection is made.

If a messaging engine does not appear to start, check if custom core group policies are being used. Verify that they are set correctly and, if preferred servers are being used, that appropriate servers are selected in the preferred server list.

Tips for troubleshooting bus members

This topic provides a set of specific tips to help you troubleshoot problems with bus members.

- Error 500 when adding a new member to a service integration bus.

Error 500 when adding a new member to a service integration bus.

When using the WebSphere administrative console to add a server as a new member of a service integration bus, you can receive an Error 500 message, with the additional message "`An error occurred while processing request: /ibm/console/sIBusMemberCollection.do`". In addition, the SystemOut log contains a null pointer exception for the stepsLayout.jsp; for example:

```
[9/21/04 14:35:18:282 CDT] 00000031 ServletWrappe E   SRVE0068E: Could not invoke
the service() method on servlet /secure/layouts/stepsLayout.jsp. Exception thrown :
java.lang.NullPointerException
```

This occurs in situations where your firewall product is configured to remove the private header information (Referer) from requests, because the WebSphere administrative console requires the header to contain the referer.

To solve this problem, you can either configure your firewall product to allow the private header information (Referer) in requests or disable the firewall product. For example, for information about configuring ZoneLabs to allow the referer in requests, see the *Private Header Information (Referer)* article at http://www.hotcomm.com/FAQ/FAQ_ZA.asp#referer.

Tips for troubleshooting the default messaging provider

This topic provides a set of specific tips to help you troubleshoot problems for JMS messaging with the default messaging provider.

For general tips about troubleshooting problems with WebSphere messaging, see Tips for troubleshooting WebSphere Messaging. This topic provides additional tips specific to the default messaging provider and its use of service integration technologies.

- JMS client applications running inside the J2EE client container fail when invoking the ConnectionFactory.createConnection method

JMS client applications running inside the J2EE client container fail when invoking the ConnectionFactory.createConnection method

When a JMS client application is running inside the J2EE client container, on a machine that is running no other WebSphere processes, a call to the ConnectionFactory.createConnection method can fail with the following error:

```
CWSIJ0005E: An instance of the channel framework service to use for communication cannot be found
```

Cause

The ConnectionFactory for the default messaging provider has a dependency on the Channel Framework Service. It locates the Channel Framework Service using a lookup in the JNDI name space. To connect to a naming service, the ConnectionFactory uses an InitialContext object created using the default constructor.

When the JMS client is running within an application server environment, the InitialContext object is able to successfully connect to the naming service, the Channel Framework Service is located and the call to createConnection completes successfully.

However, when the JMS client is executing within the J2EE client container, the InitialContext object uses the value of the java.naming.provider.url system property, to determine the location of the naming service to connect to. If no value is specified for this property, it attempts to connect to a naming service located at port 2809 on the local client machine. If there is no node agent process running on the client machine, there is no naming service listening on this port on the local machine. This causes the createConnection method to fail with the following error:

```
CWSIJ0005E: An instance of the channel framework service to use for communication cannot be found
```

Solution

A JMS client application can specify the value of the java.naming.provider.url programmatically, using code of the form:

```
String key = "java.naming.provider.url";
String value = "iiop://some.remote.machine:9810";
System.setProperty(key, value);
```

This code should be run prior to invoking the createConnection method on the ConnectionFactory object.

Alternatively, if you use the launchClient script from the command line to launch the J2EE client container, you can specify either of the following command line parameters:

- `launchClient <CLIENT EAR> -CCBootstrapHost=some.remote.machine -CCBootstrapPort=981`
- `launchClient <CLIENT EAR> -CCproviderURL=iiop://some.remote.machine:9810`

This ensures that if no provider URL is specified programmatically, then any InitialContext objects default to using the provider URL specified on the command line.

Tips for troubleshooting mediations

This topic provides a set of specific tips to help you troubleshoot problems with mediations.

To help identify and resolve problems, use the Troubleshooter reference: Messages. It contains information about error messages, indexed by message prefix. For each message there is an explanation of the problem, and details of any action that you can take to resolve the problem.

For information about looking at error logs, see Working with message logs.

Tips for common problems

- "Where is my message?"
- How do I enable trace?
- Why was my message not mediated?
- Why does my mediation not run?
- Which error messages relate to mediations?

Where is my message?

If your message is not where you expect it to be, check the following:

- The message had the reliability level `Best effort nonpersistent`, and has been discarded as a result of constrained system resources. For more information, see Reliability levels.
- The message has been received by another consumer, and has gone from the system.
- The message expiry time has exceeded, and the message has been discarded.
- The mediation returned false, and the message has been discarded. For more information, see Programming mediations .
- The mediation threw an exception, and the message was sent to the exception destination. For more information, see Error handling in mediations.
- The destination to which the message was forwarded has been deleted, and the message has been sent to the exception destination. For more information, see Exception destinations and Error handling in mediations.
- The forward routing path has been modified to behave other than intended, and the message has not been routed to the intended destination. For more information, see Configuring a destination forward routing path.
- There is an implicit loop in the forward routing path, and the message has not arrived at the intended destination.

How do I enable trace?

For more information, see Enabling trace.

Why was my message not mediated?

- Check if the destination is mediated. To do this, list the mediation points for the destination. For more information, see Listing mediation points for a bus destination.
- Look in the `system.out` log for the following error message:

  ```
  CWSIZ0002E: The mediation named {0} that is attached to destination {1} is defined to use mediation
  handler list {2}. However this handler list does not exist.
  ```

  Check for the following common reasons why the handler list does not exist:
  - The application containing the handler list has not been installed.
  - The application containing the handler list has not been started.
  - The name of the handler list is mis-spelled in the mediation definition.
- Check the status of the mediation handler application in the administrative console. If it has the status `Stopped`, check for error messages in the `system.out` log.
- Check that the message matches the selector and/or discriminator configured on the mediation. To do this:
  1. View the message on the mediation point. For more information, see Viewing messages for a bus destination.
  2. Check the configuration for the mediated destination. For more information, see Configuring mediation properties.

Why does my mediation not run?

If your mediation does not run, check the following:

- The status of the mediation in the administrative console.
- The stop reason on the mediation point. For more information, see restarting a mediation that has stopped on error.
- Error messages in `system.out` log. For more information, see Working with message logs.

Which error messages relate to mediations?

Mediations error messages have the following prefixes:

CWSJU
: Errors about message production or consumption.

CWSIZ
: Errors about what has happened to the message whilst it was mediated.

Tips for troubleshooting service integration bus security

This topic provides a set of specific tips to help you troubleshoot problems you experience when working with a secure service integration bus.

To help you identify and resolve service integration bus security-related problems, use the WebSphere Application Server trace and logging facilities as described in Enabling tracing and logging .

If you encounter a problem that you think might be related to service integration bus security, you can check for error messages in the WebSphere Application Server administrative console, and in the application server `SystemOut.log` file. You can also enable the application server debug trace to provide a detailed exception dump.

WebSphere system messages are logged from a variety of sources, including application server components and applications. Messages logged by application server components and associated IBM

products start with a unique message identifier that indicates the component or application that issued the message. The prefix for the service integration bus security component is CWSII.

For more information about the message identifier format, see the topic Message reference.

The Troubleshooter reference: Messages contains information about all WebSphere Application Server messages, indexed by message prefix. For each message there is an explanation of the problem, and details of any action that you can take to resolve the problem.

Here is a set of tips to help you troubleshoot commonly-experienced problems:
- "After you migrate an application server to WebSphere Application Server Version 6, existing Web services clients can no longer use SOAP over JMS to access services hosted on the migrated server."
- "When you try and make a connection using a user ID in an authorized group but access is denied when using LDAP" on page 242

After you migrate an application server to WebSphere Application Server Version 6, existing Web services clients can no longer use SOAP over JMS to access services hosted on the migrated server.

Before you migrated the Version 5 application server, no user ID or password was required on the target MQ Series queue. After the application server is migrated to Version 6, and using Version 6 default messaging, client requests fail because basic authentication is now enabled. The problem appears as a log message:

```
SibMessage    W   [:] CWSIT0009W: A client request failed in the application
server with endpoint <endpoint_name> in bus your_bus with reason: CWSIT0016E:
The user ID null failed authentication in bus your_bus.
```

In WebSphere Application Server Version 6, when you use the default messaging provider (service integration technologies) and WebSphere Application Server security is enabled for the server or cell, then by default the service integration bus queue destination inherits the security characteristics of the server or cell. So if the server or cell has basic authentication enabled, then the client request fails.

To resolve the problem, you have three choices (in order of security, from least secure to most secure):
- Disable security.
- For an equivalent level of security to the configuration on Version 5, modify the settings for the service integration bus that hosts the queue destination so that bus security is disabled and therefore the bus does not inherit security characteristics from the server or cell.
- For a greater level of security than the configuration on Version 5, configure basic authentication on each client that uses the service.

To disable WebSphere Application Server security, refer to either of these topics:
- Secure administration, applications, and infrastructure settings.
- Enabling and disabling administrative security using scripting.

To disable bus security, use the administrative console to complete the following steps:
1. Navigate to **Service Integration** → **Buses [Content Pane]** → **your_bus**.
2. Clear the **Secure** check box.
3. Save your changes.

To configure basic authentication on each client, use either the administrative console or the wsadmin tool. To complete the task using the wsadmin tool, see Configuring Web service client port information with the wsadmin tool and use the WebServicesClientBindPortInfo wsadmin task option. To complete the task using the administrative console, complete the following steps:

1. Navigate to **Applications** → **Enterprise Applications** → **[Content Pane]** ***application_name*** → **Web Modules or EJB Modules** → **[Content Pane]** ***module_name*** → **Web services: Client security bindings**.
2. Click **HTTP basic authentication** to access the “Configuring HTTP basic authentication with the administrative console” panel.
3. Enter the values in the panel.
4. Save your changes to the master configuration.

When you try and make a connection using a user ID in an authorized group but access is denied when using LDAP

One of the possible causes is the group name, if you are using an Lightweight Directory Access Protocol (LDAP) registry. When you specify the group authorization permissions, the distinguished name (DN) should be used as the group name. If you specify a common name (CN) for the group name users in that group cannot be authorized.

Steps to change the group name from CN to DN depends on where the problem occurred.

- If you have problem connecting to a Service Integration Bus, refer to Administering bus Connector roles, and remove any groups with CN group names, and replace them with DN group names.
- If you have problem sending a message to a destination, refer to Administering destination roles, and remove any groups with CN group names, and replace them with DN group names.
- If you have problems sending topics to a topic space, refer to Administering topic space root roles, and remove any groups with CN group names, and replace them with DN group names.
- For any other problems refer to the appropriate section on Administering authorization permissions on how to modify the group name.

Tips for troubleshooting the SIBWS

This topic provides a set of specific tips to help you troubleshoot problems you experience with the service integration bus Web services enablement (SIBWS).

To help you identify and resolve SIBWS-related problems, use the WebSphere Application Server trace and logging facilities as described in Enabling tracing and logging.

To enable trace for SIBWS, set the application server trace string to `com.ibm.ws.sib.webservices.*=all=enabled`. If you encounter a problem that you think might be related to SIBWS, you can check for error messages in the WebSphere Application Server administrative console, and in the application server `SystemOut.log` file. You can also enable the application server debug trace to provide a detailed exception dump.

A list of the main known restrictions that apply when using SIBWS is provided in SIBWS - Known restrictions.

WebSphere Application Server system messages are logged from a variety of sources, including application server components and applications. Messages logged by application server components and associated IBM products start with a unique message identifier that indicates the component or application that issued the message. The prefix for the SIBWS component is `CWSWS`.

For more information about the message identifier format, see the Message reference topic.

The Troubleshooter reference: Messages topic contains information about all WebSphere Application Server messages, indexed by message prefix. For each message there is an explanation of the problem, and details of any action that you can take to resolve the problem.

Here is a set of tips to help you troubleshoot commonly-experienced problems:
- "You need to manually install one of the SIBWS applications or resources."
- "You have a client application that works under WebSphere Application Server Version 5, but under Version 6 you get problems caused by poorly-formed requests or responses." on page 244
- "A JAX-RPC client running on WebSphere Application Server Version 5 uses SOAP over JMS to invoke a Web service running on a Version 5 application server. No user ID or password is required on the target MQ Series queue. After the application server is migrated to Version 6, and using Version 6 default messaging, client requests fail because basic authentication is now enabled." on page 244
- "You unsuccessfully attempt to create an Informix database for use with the SDO repository, and receive the message No Transaction Isolation on non-logging databases." on page 244
- "You have an inbound service that, according to the administrative console, is published to a UDDI registry. When you check the UDDI registry you discover that the service is not listed there. When you attempt to use the administrative console to republish the service to UDDI, the attempt is unsuccessful." on page 245
- "If the bus needs to pass messages through an authenticating proxy server to retrieve WSDL documents, then you must use command-line tools to retrieve the WSDL." on page 245
- "If you are using JMS to connect to a remote bus, extra configuration is required to allow Web service clients to connect to the bus." on page 245
- "You pass a large attachment through the service integration bus and you get an out-of-memory error in the Java virtual machine." on page 246
- "When you try to create a new endpoint listener configuration through the administrative console, and click New on the endpoint listener collection panel, the Please wait icon is displayed but the target panel does not appear." on page 246
- "When you try to create a new inbound service through the administrative console, the drop-down list of destinations is empty and the wizard stops you at step 1 with the error message A destination must be selected." on page 246
- "You get listener port timeout errors when large messages are passed using the SOAP over JMS endpoint listener." on page 246
- "You are getting SOAP fault messages, but cannot determine the precise problem from the fault message." on page 246
- "You are password-protecting a Web service operation, but when you install the sibwsauthbean.ear file, an error message is displayed in the WebSphere Application Server administrative console detailing a Java Naming and Directory Interface (JNDI) problem." on page 246
- "You get JNDI lookup errors when you use the same names for JMS messaging queues and queue connection factories that run on application servers on different machines." on page 247
- "You are trying to send a SOAP over HTTPS message, and you are receiving a Malformed URLException error." on page 247
- "You experience problems with handling SOAP messages with attachments." on page 247

You need to manually install one of the SIBWS applications or resources.

In WebSphere Application Server Version 6.1 and later, the following SIBWS applications and resources are installed automatically as and when needed:

- The SIBWS application (the core application that lets you configure and access Web services through a service integration bus).
- The service integration technologies resource adapter (used to invoke Web services at outbound ports).
- The endpoint listener applications (used to enable the points at which messages for inbound services are received).

For example, an endpoint listener application is installed as part of the process of creating a new endpoint listener configuration.

However if you do need to manually install one of these applications, you can still do so by using the supplied `sibwsInstall.jacl` script, and following the instructions given in the WebSphere Application Server Version 6.0 topic: SIBWS - Completing the installation.

You have a client application that works under WebSphere Application Server Version 5, but under Version 6 you get problems caused by poorly-formed requests or responses.

The service integration bus Web services enablement (SIBWS) performs more validation on Web service messages than is done in WebSphere Application Server Version 5. As a result, some client applications that use poorly-formed requests or responses (where the message parts are misnamed), and that work when using Version 5, are identified as poorly-formed in Version 6. For the steps to take to resolve the problem, see Toleration of poorly-formed SOAP messages

A JAX-RPC client running on WebSphere Application Server Version 5 uses SOAP over JMS to invoke a Web service running on a Version 5 application server. No user ID or password is required on the target MQ Series queue. After the application server is migrated to Version 6, and using Version 6 default messaging, client requests fail because basic authentication is now enabled.

The problem appears as a log message:

```
SibMessage W [:] CWSIT0009W: A client request failed in the application server with endpoint
<endpoint_name> in bus your_bus with reason: CWSIT0016E: The user ID null failed authentication
in bus your_bus.
```

For the steps to take to resolve the problem, see the following service integration technologies troubleshooting tip: “After you migrate an application server to WebSphere Application Server Version 6, existing Web services clients can no longer use SOAP over JMS to access services hosted on the migrated server.” on page 241

You unsuccessfully attempt to create an Informix database for use with the SDO repository, and receive the message “No Transaction Isolation on non-logging databases.”

If “No Transaction Isolation on non-logging databases.” is displayed as part of an exception message, it is because logging is disabled on the Informix database being used. To use an Informix database with the SDO repository, you must enable logging.

The full exception text is:

```
javax.transaction.TransactionRolledbackException: CORBA TRANSACTION_ROLLEDBACK 0x0 No; nested exception is:
    org.omg.CORBA.TRANSACTION_ROLLEDBACK: javax.transaction.TransactionRolledbackException:  ;
        nested exception is:
    javax.ejb.TransactionRolledbackLocalException:  ; nested exception is:
        com.ibm.ws.ejbpersistence.utilpm.PersistenceManagerException: PMGR1013E: Exception occured
        when verifying current backend id INFORMIX_V94: javax.resource.spi.ResourceAllocationException:
        DSRA0080E: An exception was received by the Data Store Adapter. See original exception message:
        No Transaction Isolation on non-logging db's., error code: DSA_ERROR, error code: DSA_ERROR
        vmcid: 0x0  minor code: 0  completed: No
```

Logging is disabled by default, so the create database statement `CREATE DATABASE SDOREP;` results in an exception at run time. When you create the database, use one of the following database creation statements:

- `CREATE DATABASE SDOREP WITH LOG;`
- `CREATE DATABASE SDOREP WITH BUFFERED LOG:`
- `CREATE DATABASE SDOREP WITH LOG MODE ANSI;`

You have an inbound service that, according to the administrative console, is published to a UDDI registry. When you check the UDDI registry you discover that the service is not listed there. When you attempt to use the administrative console to republish the service to UDDI, the attempt is unsuccessful.

The service was published to the UDDI registry, and the service configuration shown in the WebSphere Application Server administrative console includes a UDDI Service Key, but the service has subsequently been unpublished from UDDI without the corresponding update being applied to the WebSphere Application Server master configuration. One way in which this situation can arise is if you use the administrative console to delete an inbound service that is published to a UDDI registry, then log out of the administrative console without saving your changes. In this case, the service is unpublished from the UDDI registry, but not deleted from WebSphere Application Server (because the delete request is not confirmed, and therefore is not applied).

To update the service configuration information and republish the service to UDDI, use the administrative console to complete the following steps:

1. In the navigation pane, click **Service integration** → **Buses** → **[Content Pane]** *bus_name* → **[Services] Inbound Services** → **[Content Pane]** *service_name*. The current settings for this inbound service are displayed.
2. Click **Reload template WSDL**, then save the changes.
3. Click **Unpublish from UDDI**, then save the changes.
4. Click **Publish to UDDI**, then save the changes.

The service is successfully republished, and is reinstated in the UDDI registry.

If the bus needs to pass messages through an authenticating proxy server to retrieve WSDL documents, then you must use command-line tools to retrieve the WSDL.

Neither the administrative console panels used to create a new Web service configuration, nor the **Reload WSDL** button provided on the panels used to modify an existing Web service configuration, allow you to enter a J2C authentication alias for WSDL retrieval. Therefore when you create or modify inbound and outbound services, if the bus needs to pass messages through an authenticating proxy server to retrieve WSDL documents then you must use one of the following command-line tools to retrieve the WSDL:

- Creating a new inbound service configuration through the command line.
- Creating a new outbound service configuration through the command line.
- Refreshing the inbound service WSDL file through the command line.
- Refreshing the outbound service WSDL file through the command line.

If you are using JMS to connect to a remote bus, extra configuration is required to allow Web service clients to connect to the bus.

A Web service client application running in a server that is a member of a bus can locate a messaging engine in that bus. A Web service client application running outside of an application server - for example, running outside the WebSphere Application Server environment - cannot locate directly a suitable messaging engine to connect to in the target bus. Similarly, a Web service client application running on a server in one cell that needs to connect to a target bus in another cell cannot locate directly a suitable messaging engine to connect to in the target bus.

To enable the Web service client application to contact a target messaging engine in a remote bus, configure the JMS connection factory that the client uses so that the client can connect to a bootstrap messaging engine in the remote bus. The bootstrap messaging engine then identifies the target engine, and the information required to access the target engine is passed back to the client. For the bootstrap process to be possible, configure one or more provider end points in the connection factory used by the

client. For more information, see Configuring connection to a non-default bootstrap server.

You pass a large attachment through the service integration bus and you get an out-of-memory error in the Java virtual machine.

If this error occurs,increase the heap size as described in Tuning the SIBWS.

When you try to create a new endpoint listener configuration through the administrative console, and click New on the endpoint listener collection panel, the “Please wait” icon is displayed but the target panel does not appear.

This problem is only found with older versions of the Mozilla Web browser. Upgrade your browser to Version 1.4 or later. As a workaround, click **New** a second time and the target panel is displayed.

When you try to create a new inbound service through the administrative console, the drop-down list of destinations is empty and the wizard stops you at step 1 with the error message “A destination must be selected”.

This can only happen if there is no messaging engine on the service integration bus on which you are creating your inbound service. Create a messaging engine, then create a service destination, then re-run the wizard to create a new inbound service configuration.

You get listener port timeout errors when large messages are passed using the SOAP over JMS endpoint listener.

As with any synchronous endpoint listener, timeout errors can occur. To minimize the frequency of timeout errors, increase the timeout settings for the endpoint listener. If the problem persists, then disable trace and logging for service integration technologies by setting the application server trace string to `com.ibm.ws.sib.webservices.*=all=disabled`.

You are getting SOAP fault messages, but cannot determine the precise problem from the fault message.

If you receive a SOAP fault message with a faultstring that is just the value of one of the parameters of the invocation, that means the parameter value is not valid. For example if you have a service that expects an `int` parameter and you send it a message containing the value “1.1”, then the fault message you receive contains `1.1` as the fault string:

```
<faultcode>SOAP-ENV:Client</faultcode>
<faultstring>1.1</faultstring>
```

This message is consistent with Apache SOAP behavior, and is not correctable by the service integration technologies.

You are password-protecting a Web service operation, but when you install the sibwsauthbean.ear file, an error message is displayed in the WebSphere Application Server administrative console detailing a Java Naming and Directory Interface (JNDI) problem.

When you password-protect a Web service operation, check that you enter, in the “EJB References” for the authorization session bean, the correct JNDI name of the imported Web service enterprise bean. Note that this home is case sensitive.

You get JNDI lookup errors when you use the same names for JMS messaging queues and queue connection factories that run on application servers on different machines.

You should not use the same names for messaging queues and queue connection factories that run on application servers on different machines, because the service integration technologies always look first for JMS destinations locally, and only use the full JNDI reference if they cannot find the destination locally. If you configure as an outbound service a Web service that is hosted on a remote machine, and you use the same names for messaging queues and queue connection factories on the remote machine and on the machine on which the outbound service is hosted, then the service integration technologies find and use the local queues even if the remote JNDI destination is provided in full in the WSDL service definition.

You are trying to send a SOAP over HTTPS message, and you are receiving a Malformed URLException error.

The service integration technologies can use Secure Sockets Layers (SSL) to invoke external Web services that include `https://` in their addresses. For more information see Invoking outbound services over HTTPS.

You experience problems with handling SOAP messages with attachments.

See You pass a large attachment through the service integration bus and you get an out-of-memory error in the Java virtual machine..

Tips for troubleshooting WS-Notification

To help you identify and resolve WS-Notification-related problems, use the WebSphere Application Server trace and logging facilities as described in Enabling tracing and logging.

To enable trace for WS-Notification, set the application server trace string to `SIBWsn=all=enabled:com.ibm.ws.sib.webservices.*=all=enabled`. If you encounter a problem that you think might be related to WS-Notification, you can check for error messages in the WebSphere Application Server administrative console, and in the application server `SystemOut.log` file. You can also enable the application server debug trace to provide a detailed exception dump.

A list of the main known restrictions that apply when using WS-Notification is provided in WS-Notification - known restrictions.

WebSphere Application Server system messages are logged from a variety of sources, including application server components and applications. Messages logged by application server components and associated IBM products start with a unique message identifier that indicates the component or application that issued the message. The prefix for the WS-Notification component is `CWSJN`.

For more information about the message identifier format, see the Message reference topic.

The Troubleshooter reference: Messages topic contains information about all WebSphere Application Server messages, indexed by message prefix. For each message there is an explanation of the problem, and details of any action that you can take to resolve the problem.

Here is a set of tips to help you troubleshoot commonly-experienced problems:

- “Exception caused by incorrect configuration of the SDO repository.” on page 248
- “Multiple messages received by a notification consumer for each event notification” on page 248
- “Deleting administered subscribers and messaging engines” on page 248
- “Use of Web service destinations” on page 249
- “Failure of an inbound (application to broker) notification” on page 249

Exception caused by incorrect configuration of the SDO repository.

If you try to create a WS-Notification service, and you get the following stack trace, then SDO repository is not configured correctly. To resolve this problem, see Installing and configuring the SDO repository.

```
java.lang.Exception: com.ibm.ws.sib.webservices.admin.config.SIBConfigException: CWSWS5010E:
Failed to store WSDL located at http://www.ibm.com/websphere/wsn/notification-broker
due to the following exception: com.ibm.ws.sib.webservices.exception.SIBWSUnloggedException:
CWSWS1007E: The following exception occurred:
com.ibm.ws.sdo.config.repository.impl.RepositoryRuntimeException:
javax.transaction.TransactionRolledbackException: CORBA TRANSACTION_ROLLEDBACK 0x0 No;
nested exception is: org.omg.CORBA.TRANSACTION_ROLLEDBACK:
javax.transaction.TransactionRolledbackException: ; nested exception is:
javax.ejb.TransactionRolledbackLocalException: ; nested exception is:
com.ibm.ws.ejbpersistence.utilpm.PersistenceManagerException:
PMGR1014E: Exception occured when getting connection factory:
com.ibm.websphere.naming.CannotInstantiateObjectException:
threw NameNotFoundException while the JNDI NamingManager was processing a
javax.naming.Reference object. [Root exception is javax.naming.NameNotFoundException:
Context: smeagolNode03Cell/nodes/smeagolNode03/servers/server1, name:
jdbc/com.ibm.ws.sdo.config/SdoRepository:
First component in name com.ibm.ws.sdo.config/SdoRepository not found.
[Root exception is org.omg.CosNaming.NamingContextPackage.NotFound:
IDL:omg.org/CosNaming/NamingContext/NotFound:1.0]] vmcid: 0x0 minor code: 0 completed: No.
```

Multiple messages received by a notification consumer for each event notification

In some situations you might receive more notifications at a given notification consumer than the number of event notifications that have been inserted into the notification broker by a publisher. For example you might publish 4 messages, and receive 8, 12, 16 (or some other multiple of four) messages at the notification consumer.

This is normally caused by there being two or more active subscriptions that target the notification consumer - a situation that can occur if the subscriber application is run more than once. Each time the Subscribe operation is called, a new subscription must be created by the notification broker (see section 4.2 of the Web Services Base Notification specification), which causes duplicate messages to be delivered if a previous subscription exists.

To check whether this is what is happening, examine the **SubscriptionReference** property of the notifications received by the notification consumer. This endpoint reference contains the identifier of the subscription that caused the notification to be sent. If you find several different subscription identifiers, then there is more than one subscription active.

Subscriber applications should tidy up subscriptions when they are not required (or register them with a timeout), however you can tidy them up administratively using the run-time panels as described in Listing or deleting active WS-Notification subscriptions.

Deleting administered subscribers and messaging engines

Deleting administered subscribers

You should be wary of deleting and recreating messaging engines on bus members for which WS-Notification administered subscribers have been configured, because in some cases this can leave the remote Web service subscription active (and passing notification messages to the local server) even though there is no longer any record of it.

To avoid this situation you should delete the WS-Notification configuration, or just the administered subscribers, in a separate step to deleting the messaging engine. When the dynamic configuration update is then processed, or the server restarted, the remote Web service subscription is tidied up cleanly.

Note: This problem does not occur if it is only the WS-Notification configuration that is modified; it only occurs if the messaging engine is also deleted.

Use of Web service destinations

When a WS-Notification service is created, three inbound services are configured for the WS-Notification service, one for each of the three WS-Notification service roles:

- Notification broker
- Subscription manager
- Publisher registration manager

These inbound services are defined on the same service integration bus as the WS-Notification service, and each of these inbound services refers to the same bus destination. Usually, a bus destination can be used as described in Learning about bus destinations. However, this is not the case for WS-Notification. This destination does not relate to the topics for which the WS-Notification service can handle requests and you should not alter or mediate the destination. In WS-Notification, the configuring of topics is handled through topic namespaces. For more information, see Creating a new permanent WS-Notification topic namespace.

Failure of an inbound (application to broker) notification

Applications wishing to publish event notifications into the broker make use of the Notify operation. This is defined as a one-way (Web services) operation which means that it is not possible to return a fault (exception) if it is not possible to complete the operation. Thus the application will assume that the notification was successful, but subscribing applications will not receive the notification message.

The cause of this type of failure might be an application error (invalid topic syntax), or a mismatch between the application code and the server configuration (using an undefined topic namespace). Specific reasons for which an inbound notification might fail include the following:

- Invalid topic: the topic expression supplied does not match the syntax of the stated dialect, or they specified an unsupported dialect. This is an application error.
- Invalid topic namespace: the application specifies a topic namespace that has not been configured, but the administrator has specified on the WS-Notification service that we should not permit use of dynamic namespaces. This is caused by a mismatch between the application and the WS-Notification Service topic namespaces.
- Unsupported topic: the specified topic is prohibited by a topic namespace document that has been applied to the topic namespace (for example it is a sub-topic of a topic that has been marked as “final”).
- Invalid credentials: the specified user ID or password is not valid or does not have the required authority. This is caused by a mismatch between the application configuration and the server security policies.
- Publisher not registered: The application tries to send a notification without first registering with the broker, but the administrator has configured the WS-Notification service to require registration of applications. This is caused by an application error - a well behaved application should first check whether it is required to register before publishing to a broker.
- The associated service integration bus topic space has been disabled.

You need to monitor this type of failure closely, because it might indicate a denial of service attack and certainly indicates that the application is not functioning correctly. The first time an inbound notification fails from a particular producing application, a warning message is sent to the SystemOut log of the server. If

there are further notification failures for that producer, subsequent timed warning messages are logged at 30 minute intervals. Additional information is provided with each timed message to indicate how many failed notifications were received for that producer during the 30 minute interval.

When the system generates each warning message, it identifies the producing application through one of two identifiers:

- The ProducerReference element of the NotifyMessage provided in the Notify operation . This element uniquely identifies the application. However this element is optional.
- The IP address of the host that originated the request. This address might not uniquely identify the application, but it narrows your search.

Note: The system cannot identify the host IP address in all cases. For example, for SOAP over JMS transports the IP address of the originating host is not available or applicable.

Failure of an outbound (broker to application) notification

The failure of an outbound Web service invocation (broker to application) is caused when a remote application is unavailable for invocation, and might be the result of an application failure, a network error, or a firewall configuration issue. Failure to pass event notifications to subscribed applications causes messages to build up on the subscriptions held on the server. The messages held on a given subscription can be observed using the run-time panels as described in Listing or deleting active WS-Notification subscriptions. Subscriptions for which the most recent event notification attempt has failed in this way are marked as being in **ERROR** state when viewed in the WS-Notification subscription runtime administration panel.

If the WS-Notification service point fails to successfully notify a NotificationConsumer application, a warning message is sent to the application server SystemOut log and the subscription is told to wait for 2 minutes. Reasons for a failure of this type might be that the remote Web service is not currently available, or that network conditions prevent contact between the local server and the service.

After 2 minutes, the notification is retried. If delivery is still not possible then the subscription is put back into a wait state for another 2 minutes. If the failure is caused by a transient I/O error, this pattern is repeated indefinitely, until the notification is either successfully delivered or you delete the subscription. If the error is caused by an application failure on the remote side then the notification will be retried up to the number of times defined in the 'Maximum failed deliveries' setting of the service integration bus topic space destination from which the message is being received. After the first warning message is output to the SystemOut log, subsequent timed warning messages are logged at 30 minute intervals.

Tidying up stateful resources that are not automatically deleted

The act of subscribing to the broker or registering a publisher creates a stateful resource on the server that consumes system resources while it is active. Normally an application specifies a termination time as part of the act of creating these resources, and thus they are automatically deleted when the termination time is reached. However it is also possible for the application to request an infinite lifetime for the resource. If this is done then it is possible for resources to remain on the server indefinitely even though the application might never be coming back to use (or destroy) the resource.

You can to view the stateful resources (subscriptions and publisher registrations) using the run-time panels described in Interacting at run time with WS-Notification. These panels also provide the ability to administratively delete the items if required. Only do this if you are sure that the application is no longer using the resource because it will cause application failures if the resource is referenced after being deleted.

Failure to delete a durable subscription that was created by WS-Notification, when using the service integration bus panel

When you create a subscription using a WS-Notification application, in other words by using the Subscribe operation, one or more durable subscriptions are created in the relevant service integration bus topic space destination. You can view these durable subscriptions in the service integration bus runtime panels for the publication point.

The runtime panels for the publication point also provide the ability to delete one or more durable subscriptions. However, if you use this function to delete a subscription that was created by a WS-Notification application, the delete operation fails. This failure occurs because the WS-Notification implementation maintains an active consumer for this durable subscription for the duration of the time that the server is running, and a durable subscription cannot be deleted if an active consumer is present.

Note: This deletion restriction also applies to durable subscriptions created by other applications, such as JMS applications.

To delete a subscription that was created by a WS-Notification application, use the runtime panels provided by the WS-Notification implementation, as described in Interacting at run time with WS-Notification. This approach closes the active consumer and automatically deletes the related service integration bus durable subscriptions.

Administrative stop of a messaging engine

WebSphere Application Server depends on being able to access a running service integration bus messaging engine to send and receive messages, and to create and retrieve state for the various Web service resources that are created.

You can stop a messaging engine using the MBean interface or run-time panels. This prevents WS-Notification from successfully servicing any requests from applications that might come in during the time that the messaging engine is stopped. In this situation, error messages are logged as described in “Failure of an inbound (application to broker) notification” on page 249 and “Failure of an outbound (broker to application) notification” on page 250. When you stop a messaging engine, all WS-Notification processing stops and all messaging applications cease to function. When you restart the messaging engine, WS-Notification processing resumes.

Failures as a result of changes in topic space and topic namespace configurations

The WS-Notification configuration artefacts often depend on objects defined in other areas of the server configuration. For example the endpoint listeners through which application requests are received, and the service integration bus topic spaces to and from which messages are sent.

The following items describe the action that is taken by the WS-Notification run-time code when it meets relevant changes in the objects upon which it depends.

Deleting a service integration bus topic space

The service integration bus topic space is the primary messaging object upon which WS-Notification depends at run time. Notification messages from an application are published to the topic space specified by the (permanent) topic namepace mapping specified by the administrator.

Deleting a service integration bus topic space has the following effects upon new and existing WS-Notification applications:

- RegisterPublisher requests using a WS-Notification topic namespace that references the deleted topic space receive a TopicNotSupportedFault error message.

- Notify requests for a topic associated with the deleted topic space do not publish the message to the topic space (because it has been deleted). The application is not informed because no faults are thrown by the Notify operation (see “Failure of an inbound (application to broker) notification” on page 249).
- Subscribe requests using a WS-Notification topic namespace that references the deleted topic space receive a SubscribeCreateFailedFault error message.
- No further messages are delivered to applications that have existing subscriptions to the deleted topic space. The existing subscription is deleted, and any attempt to invoke operations on the subscription (for example getCreationTime) results in a ResourceUnknownFault error message.
- Deleting and recreating a service integration bus topic space is considered as two separate steps. Existing subscriptions are deleted in response to the first step, and therefore do not exist when the topic space is recreated.

Deleting a permanent topic namespace mapping

Deleting the topic namespace mapping that was used to establish a (currently active) subscription has the same effect as deleting the underlying service integration bus topic space as defined previously, and subscriptions that were created using this namespace mapping are deleted.

Publisher registrations and pull points associated with the deleted topic namespace mapping are also deleted.

Changing a permanent topic namespace mapping

The fields of a permanent topic namespace mapping are read-only fields, so the only way to “change” the fields is to delete the namespace mapping and recreate it with new values. The effect of deleting a permanent topic namespace mapping is described in the previous item.

Failure to create a WS-Notification service without a configured SDO repository

During the process of creating a WS-Notification service it is necessary for WSDL documents to be saved into the SDO repository. You will see the following error message if you attempt to create a WS-Notification service through the administrative console, or through scripting, before successfully configuring the SDO repository.

```
java.lang.Exception: com.ibm.ws.sib.webservices.admin.config.SIBConfigException: CWSWS5010E: Failed to
store WSDL located at http://www.ibm.com/websphere/wsn/notification-broker due to the following
exception:
    com.ibm.ws.sib.webservices.exception.SIBWSUnloggedException: CWSWS1007E: The following exception
        occurred: com.ibm.ws.sdo.config.repository.impl.RepositoryRuntimeException:
        javax.transaction.TransactionRolledbackException: CORBA TRANSACTION_ROLLEDBACK 0x0 No; nested
        exception is: org.omg.CORBA.TRANSACTION_ROLLEDBACK:
        javax.transaction.TransactionRolledbackException: ;
    nested exception is: javax.ejb.TransactionRolledbackLocalException: ;
    nested exception is: com.ibm.ws.ejbpersistence.utilpm.PersistenceManagerException: PMGR1014E:
        Exception occured when getting connection factory:
        com.ibm.websphere.naming.CannotInstantiateObjectException: threw NameNotFoundException while
        the JNDI NamingManager was processing a javax.naming.Reference object.
    [Root exception is javax.naming.NameNotFoundException: Context:
        KADGINNode01Cell/nodes/KADGINNode01/servers/server1, name:
        jdbc/com.ibm.ws.sdo.config/SdoRepository: First component in name
        com.ibm.ws.sdo.config/SdoRepository not found.
    [Root exception is org.omg.CosNaming.NamingContextPackage.NotFound:
        IDL:omg.org/CosNaming/NamingContext/NotFound:1.0]] vmcid: 0x0 minor code: 0 completed: No.
```

For details on how to configure the SDO repository, see Installing and configuring the SDO repository.

Tips for troubleshooting WebSphere MQ link

This topic provides a set of specific tips to help you troubleshoot problems with WebSphere MQ link.

- Use this topic if channels fail to start.
- Use this topic if messages are not delivered.
- Use this topic if an application server cannot shutdown if a WebSphere MQ link sender channel is waiting for its disconnect interval to expire.

Channels do not start

Note: Error messages appear in the SystemOut.log file or, if you have turned on tracing, in the trace.log file.

1. Verify that the channel names specified on the WebSphere MQ link sender channel and/or the MQLinkReceiver definitions match those specified on the sender and/or receiver channel definitions in the WebSphere MQ network. Channel names are case sensitive.
2. Verify that the channel sequence numbers are not out of step. If they are, then the channel will remain in a state of retry until the sequence numbers have been reset. The sequence numbers are reset by WebSphere MQ. On Windows, if using the WebSphere MQ Explorer, you can right click on the channel and select All Tasks>Reset.

   Look for messages `CWSIC3011E`, `CWSIC3015E`.
3. Verify that both ends of the channel have been defined and configured correctly. It is possible that the channel at the remote end is currently in a stopped state and therefore is currently unavailable. Start the channel at the remote end if possible.

   Look for messages `CWSIC3018E`, `CWSIC3113E`, `CWSIC3114E`, `CWSIC3236E`.
4. Verify that the channel sequence number wrap values are the same at both ends of the channel.

   Look for message `CWSIC3010E`.
5. Verify that the WebSphere MQ link sender channel or the sender channel is not in an in doubt state. Resolve the channel if required. The channel is resolved by WebSphere MQ. On Windows, if using the WebSphere MQ Explorer, you can right click on the channel and select All Tasks>Resolve.

   Look for message `CWSIC3065E`.
6. Verify that the listeners have been started, and are listening on the correct ports. By default, service integration listens on port 5558 for inbound connections while the WebSphere MQ network listens on port 1414.

Messages are not delivered

Note: Error messages appear in the SystemOut.log file or, if you have turned on tracing, in the trace.log file. You can also look for equivalent messages in the WebSphere MQ error logs (or trace files if you have turned on tracing in the WebSphere MQ network).

1. If you are sending messages from a service integration bus to a WebSphere MQ network, it is possible that the messages are stored on the service integration bus and waiting to be delivered, but that the WebSphere MQ link sender channel channel has not been started or is in a retry state.

   Verify that the WebSphere MQ link sender channel is started and in running state.
2. If you are sending messages from a WebSphere MQ network to a service integration bus, it is possible that the messages are stored on the transmission queue in the WebSphere MQ network and waiting to be delivered, but that the sender channel in the WebSphere MQ network has not been started or is in a retry state.

   Verify that the sender channel in WebSphere MQ network is started and in running state.
3. It is possible that the messages could not be processed or delivered to the target destination and hence they have been placed either on an exception destination on the service integration bus, or on the dead letter queue in the WebSphere MQ network. Verify that the WebSphere MQ Link on the messaging engine is configured properly with the correct foreign bus, queue manager name (service

integration bus), sender channel and receiver channel. The sender channel on the WebSphere MQ Link should match the receiver channel on WebSphere MQ. The receiver channel on the WebSphere MQ Link should match the sender channel on WebSphere MQ.

Look for messages CWSIC3096I, CWSIC3098I, CWSIC3200E, CWSIC3209E.

Check the exception destination(s) and the dead letter queue. It is possible that the target destination has not been defined, or is full in which case, determine why messages are not being processed from the target destination.

4. It is possible that the target destination and the exception destination and/or the dead letter queue are full and that subsequent persistent messages cannot be safely delivered. Under these circumstances the channel is stopped to avoid any loss of messages.

   Look for message CWSIP0291W.

   Determine why messages are not being processed from the target destination.
5. It is possible that the target destination and the exception destination and/or dead letter queue are full and that subsequent nonpersistent messages are discarded.

   Check the persistence of messages being generated by your application(s).
6. It is possible that the channel has stopped because the remote system cannot accept messages for some reason.

   Look for message CWSIC3080E.

An application server cannot shutdown if a WebSphere MQ link sender channel is waiting for its disconnect interval to expire

If a WebSphere MQ link sender channel does not have any messages to deliver, it waits for its specified disconnect interval before timing out. If the application server is shut down while a WebSphere MQ link sender channel is in a wait state, the application server waits for the WebSphere MQ link sender channel to timeout before shutting down. A long disconnect interval might delay the server shutdown.

If the application server shutdown is delayed by a WebSphere MQ link sender channel in a wait state, you have two options:

- Attempt to put a message onto the transmission item stream for the WebSphere MQ link sender channel. Note that this might not take the channel out of its wait state if the application server shutdown is already in progress
- Force the termination of the application server process.

To reduce possible delays during application server shutdown, you can specify a smaller value for the disconnect interval. Note that a discount interval of 0 indicates an indefinite wait. For more information about setting the disconnect interval for a WebSphere MQ link sender channel, see Modifying a WebSphere MQ link sender channel.

Troubleshooting service integration message problems

This topic contains links to tasks that will help you to investigate problems with messages, such as messages not arriving or being consumed, or poison messages.

If you are having problems with messages not behaving as you expect, use the links below to navigate to the topic that is appropriate for your problem.

- “Understanding why best effort messages are being discarded” on page 255
- “Investigating why a queue is full” on page 255
- “Investigating why a topic space is full” on page 256
- “Investigating why point-to-point messages are not arriving” on page 258
- “Investigating why point-to-point messages are not being consumed” on page 263
- “Investigating why publish/subscribe messages are not arriving at a subscription” on page 270

Understanding why best effort messages are being discarded

A reliability level of *best effort* means that messages might be lost during normal functioning of the system, for example if the connection used to send the messages is busy. Although this is normal and expected, you might want to investigate the reasons for messages being lost.

Use this task when best effort messages are being discarded by a running system, and you want to understand the possible causes. For more information about 'best effort' and other message reliability levels, see Message reliability levels.

The following list explains some of the reasons for losing best effort messages:

- The destination queue or topic space is already full to a level higher than the high message threshold. To check whether this is the case, click **Service integration -> Buses -> *bus_name* ->** [Additional Properties] **Destinations -> *destination_name*** and under **Message points** click the relevant point type (for example, **Queue points**). Click the relevant message point to display its general properties, and compare the values of the **High message threshold** and **Current message depth** fields.
- The connection to the target system is down.
- The connection to the target system is busy. Any best effort messages that cannot be sent will be discarded.
- The system in general is busy, for example a messaging engine might be occupied processing higher reliability messages for another destination.
- There is a temporary network problem. Look in the error log for more information.

Investigating why a queue is full

This topic describes how to investigate why a queue on a service integration bus is full.

When a queue becomes full, exceptions will be returned when you attempt to produce a message to that queue. The most probable reason for a queue filling up is that the producing application is producing messages faster than they can be consumed by the consuming application, although causes can also include broken communication links or errors in the consuming application.

1. Click **Service integration -> Buses -> *bus_name* -> Destinations** to display a list which includes all the queues on that bus. Click the name of the queue that is full.
2. Click **Queue points -> *queue_point_name***, then on the **Runtime** tab review the value of the **Current message depth**. If this value increases steadily, the producing application is outpacing the consumer.

   Note: If the destination has multiple queue points, or is mediated, perform the following checks for each message point the message could have been sent to or consumed from.
3. Determine which messaging engines the producing and consuming applications are connected to, see "Determining which messaging engine an application is connected to" on page 256.
4. If the producing and consuming applications are connected to different messaging engines, the messages are being routed via a remote queue point. On the producer's messaging engine, click **Remote queue points** and then click the queue point that represents the consumer's queue point. Review the number of current outbound messages. If the number of current messages is low, the problem does not lie with the remote queue point; check that the consuming application is started and is consuming messages without error. If the number of current messages is approaching the high message threshold, perform the following checks:
   - Check that the two messaging engines can communicate with each other, see "Service integration troubleshooting: Checking the communication between two messaging engines in a bus" on page 256. If the messaging engines can communicate, reduce the rate at which messages are produced. If the messaging engines cannot communicate, resolve the failure. If you encounter problems processing the backlog of messages once communication is restored, and the backlog does not

contain any messages that are vital, consider deleting all the messages on the remote message point. To delete the messages, select the relevant remote message point and click **Delete all messages**.

Note: You will not be able to recover the messages once they have been deleted.

- Check that messages are not being trapped in the 'Committing' state. If they are, a resource manager, such as a database, has hung. Resolve the issue with the resource manager. If this fails, note the **Transaction ID** of the message and click **Servers -> Application servers ->** [Content pane] ***server_name*** **->** [Runtime tab] [Additional Properties] **Transaction Service** to display the general properties for the transaction service, including numbers of transactions. Use the **Review** links to resolve the transaction whose **Global ID** matches the transaction ID of the message.

Determining which messaging engine an application is connected to

If your application fails to receive or produce a message, you might want to find out which messaging engine it is connected to, as part of troubleshooting the problem.

1. If your application is a JMS application, examine its connection factory as the messaging engine name might be specified there.
2. If your application is not a JMS application, or its connection factory does not specify the messaging engine name, use one of the following methods to determine which messaging engine the application is connected to:
   - Within the application code, after the application has obtained a valid Connection object, add a call to the toString() method of that object. The connected messaging engine name will be clearly listed when you rerun the application.
   - Enable the SIBJms_External trace component and rerun the application. Inspect the generated trace for a reference to the connected messaging engine name.

Be aware that the messaging engine name returned by either of these methods relates to the rerun of the application. It is possible that the original failing instance of the application was connected to a different messaging engine in the bus.

Service integration troubleshooting: Checking the communication between two messaging engines in a bus

If you are troubleshooting a problem with your service integration system, you might want to check that two messaging engines can communicate with each other.

1. Check that both messaging engines are running.
2. For each messaging engine:
   a. Check the system log for a CWSIT0028I message that relates to the two messaging engines in question. This message indicates that the two messaging engines are successfully communicating with each other.
   b. Find the most recent instance (there may be only one) of the CWSIT0028I message for the two messaging engines, and check the log to make sure that a CWSIT0029I message for these two messaging engines does not appear later in the log. This message indicates that the communication connection between the two messaging engines has failed.

   If either of these checks fails, inspect the log for indications of the cause of the failure, and follow the suggested actions to rectify the problem.

Investigating why a topic space is full

This topic describes how to investigate why a topic space on a service integration bus is full.

When a topic space becomes full, exceptions will be returned when you attempt to publish a message to that topic space. The most probable reason for a topic space filling up is that the publishing application is producing messages faster than they can be consumed by the subscribing application or applications. However there could be other causes, such as dormant subscribers or broken communications links.

Another possible cause is a regular increase in message traffic, for example at certain times of day. Consider increasing the high message threshold to overcome this problem.

1. Click **Service integration -> Buses -> *bus_name* -> Destinations** to display a list which includes all the topic spaces on that bus. Click the name of the topic space that is full.
2. Click **Publication points -> *publication_point_name***, then on the **Runtime** tab review the value of the **Current message depth**. If this value increases steadily, the publishing application is outpacing the subscribers. Click **Subscriptions** to display the subscriptions for the topic space. For each subscription, click on the subscription name and examine the **Current message depth**. If all the subscriptions are filling up, reduce the rate at which the publishing application is publishing messages.

   Note: If the topic space is mediated, perform the following checks for each mediation point the message could have been sent to or consumed from.
3. If only one subscription is filling up, the problem lies with the related subscribing application. If the subscription is nondurable, modify the subscribing application to increase the speed of consumption.
4. If the subscription is a durable subscription, click **Messages** and ensure that the message at the top of the list changes with time; this indicates that the subscribing application is actually consuming messages. If the message does not change but the application is running, either delete the subscription or increase the high message threshold of the publication point.
5. Determine which messaging engines the publishing and subscribing applications are connected to, see “Determining which messaging engine an application is connected to” on page 256.
6. If the publishing and subscribing applications are connected to different messaging engines, the messages are being routed via a remote queue point. On the publisher’s messaging engine, click **Remote publication points** and then click the publication point that represents the subscriber’s publication point. Review the number of current outbound messages. If the number of current messages is low, the problem does not lie with the remote message point. If the number of current messages is approaching the high message threshold, perform the following checks:
   - Check that the two messaging engines can communicate with each other, see “Service integration troubleshooting: Checking the communication between two messaging engines in a bus” on page 256. If the messaging engines can communicate, reduce the rate at which messages are published. If the messaging engines cannot communicate, resolve the failure. If you encounter problems processing the backlog of messages once communication is restored, and the backlog does not contain any messages that are vital, consider deleting all the messages on the remote message point. To delete the messages, select the relevant remote message point and click **Delete all messages**.

     Note: You will not be able to recover the messages once they have been deleted.
     To avoid the messages building up again, click **Topics**, then click **Clear all**. No more messages will be sent to this remote publication point. To reset the topic list, restart the messaging engine.
   - Check that messages are not being trapped in the ’Committing’ state. If they are, a resource manager, such as a database, has hung. Resolve the issue with the resource manager. If this fails, note the **Transaction ID** of the message and click **Servers -> Application servers ->** [Content pane] ***server_name* ->** [Runtime tab] [Additional Properties] **Transaction Service** to display the general properties for the transaction service, including numbers of transactions. Use the **Review** links to resolve the transaction whose **Global ID** matches the transaction ID of the message.

Determining which messaging engine an application is connected to

If your application fails to receive or produce a message, you might want to find out which messaging engine it is connected to, as part of troubleshooting the problem.

1. If your application is a JMS application, examine its connection factory as the messaging engine name might be specified there.
2. If your application is not a JMS application, or its connection factory does not specify the messaging engine name, use one of the following methods to determine which messaging engine the application is connected to:

- Within the application code, after the application has obtained a valid Connection object, add a call to the toString() method of that object. The connected messaging engine name will be clearly listed when you rerun the application.
- Enable the SIBJms_External trace component and rerun the application. Inspect the generated trace for a reference to the connected messaging engine name.

Be aware that the messaging engine name returned by either of these methods relates to the rerun of the application. It is possible that the original failing instance of the application was connected to a different messaging engine in the bus.

Service integration troubleshooting: Checking the communication between two messaging engines in a bus

If you are troubleshooting a problem with your service integration system, you might want to check that two messaging engines can communicate with each other.

1. Check that both messaging engines are running.
2. For each messaging engine:
   a. Check the system log for a CWSIT0028I message that relates to the two messaging engines in question. This message indicates that the two messaging engines are successfully communicating with each other.
   b. Find the most recent instance (there may be only one) of the CWSIT0028I message for the two messaging engines, and check the log to make sure that a CWSIT0029I message for these two messaging engines does not appear later in the log. This message indicates that the communication connection between the two messaging engines has failed.

   If either of these checks fails, inspect the log for indications of the cause of the failure, and follow the suggested actions to rectify the problem.

Investigating why point-to-point messages are not arriving

This topic describes how to investigate why point-to-point messages are not arriving at a destination on a service integration bus.

Use this topic if you have an application that is producing point-to-point messages in a service integration system, and the messages are not arriving at their destination.

Perform the following preliminary checks before starting the investigation:

- Check that the producing application is producing messages correctly:
  - Check that there are no failures in the application.
  - Check that the name of the destination is correct.
  - Check that the transaction used to produce the message was committed without any exceptions.
  - Check that the application is allowing sufficient time for messages to be delivered; messages are transmitted asynchronously between messaging engines, so if the messages are being routed through a remote message point there may be a slight delay before they are delivered. The length of the delay is dependent upon factors such as system capacity and loading.
- Check the producing application to see if it is giving the messages a short expiry time. If this is the case, the messages may be expiring before they arrive, or before they can be processed by the receiving messaging engine.
- Examine the relevant exception destination to see if the messages appear there. If they do, use the information contained within the messages to understand why they have arrived at the exception destination, and write an application (or mediation) to process the messages.
- Check the reliability of the messages. If the reliability is set to best effort, the messages can be discarded by the system during normal operation. See “Understanding why best effort messages are being discarded” on page 255 for a list of possible causes.

- Check the priority of the messages. If the priority is low, higher priority work may be delaying the messages.
- Check the system environment, for example, a busy CPU might cause delayed messages.
- Examine the error logs for exceptions.

1. Click **Service integration -> Buses -> *bus_name* ->** [Destination resources] **Destinations** to display the destinations on the relevant bus. Click on the destination and check that the **Send allowed** check box is selected.
2. Stop the consuming application. Clear the **Receive allowed** check box for the destination and save the changes to the master repository. If you do not have dynamic configuration enabled, restart the messaging engine for the changes to take effect. This will prevent any consumers from consuming the test message that you will use to investigate the problem.
3. Run the producing application to produce a test message with a reliability level greater than best effort (best effort messages can be discarded during normal operation so are not useful for investigating this problem). The following steps describe how to investigate what happens to the test message.
4. Determine which messaging engine hosts the queue point for the destination to which the messages are being sent, see "Determining the location of message points for a destination on a service integration bus."
5. Click **Service integration -> Buses -> *bus_name* ->** [Topology] **Messaging engines** and check that the messaging engine is running.
6. From the messaging engine panel click [Message points] **Queue points ->*queue_point_name*** [Runtime tab] **Messages** to view the messages on the queue point. If the message is displayed, it arrived successfully at the messaging engine and the problem has cleared.
7. Determine which messaging engine the producing application is connected to, see "Determining which messaging engine an application is connected to" on page 256.
8. If the producing application is connected to a messaging engine other than the messaging engine hosting the queue point, the messages are being routed through a remote message point. Refer to "Investigating why point-to-point messages are not arriving through a remote message point" on page 260 to investigate this scenario.

Determining the location of message points for a destination on a service integration bus

As part of troubleshooting a problem, you might need to find where message points for a destination are located.

Perform this task as part of problem determination, when you need to find out which messaging engine a message point is located on.

Note: If you have an alias destination, the alias is resolved to the physical destination immediately after a message is produced. Follow the steps below for the physical destination.

Click **Service integration -> Buses -> *bus_name* ->** [Additional Properties] **Destinations** to display the destinations on the relevant bus. Review the **Type** of the destination:

- If the destination is a queue, the number of queue points depends on whether the queue is localized to a single messaging engine, or a cluster of messaging engines. In either case the name of the queue point or points is of the form *destination@messaging_engine_name*:
  - If the queue is localized to a single, non clustered messaging engine, there will be a single queue point.
  - If the queue is localized to a cluster, there will be one queue point for every messaging engine in the cluster.

  If the queue is mediated it will have one or more mediation points (depending on whether it is localized to a single messaging engine or a cluster). Messages produced to a mediated destination will first be

delivered to a mediation point, processed by the mediation or mediations, and unless routed otherwise, finally be delivered to one of the queue's queue points.

- If the destination is a topic space, it will have a publication point localized to every messaging engine in the bus. Messages produced to a topic space are always delivered directly to the publication point situated on the same messaging engine that the producing application is connected to. A topic space may be mediated in the same way as a queue, in which case messages will initially be directed to the one or more mediation points and then to a publication point colocated with the mediation point.

Continue with the problem determination. If the destination is a queue with multiple queue points, or it is mediated using multiple mediation points, perform the problem determination for each message point that the message could have been sent to or consumed from.

Investigating why point-to-point messages are not arriving through a remote message point

This topic describes how to investigate why point-to-point messages are not arriving at a destination on a service integration bus, when the messages are being routed through a remote message point.

Follow the steps in "Investigating why point-to-point messages are not arriving" on page 258, which contains preliminary checks and investigative tasks that you should carry out before proceeding with this task.

You should perform this task as part of "Investigating why point-to-point messages are not arriving" on page 258. This task explains how to investigate the flow of messages in a point-to-point messaging scenario where the messages are being routed through a remote message point. The following diagram illustrates the situation. ME1 is the messaging engine that the producing application is attached to, and ME2 is the messaging engine that is hosting the queue point. These messaging engines are referred to in the following steps.

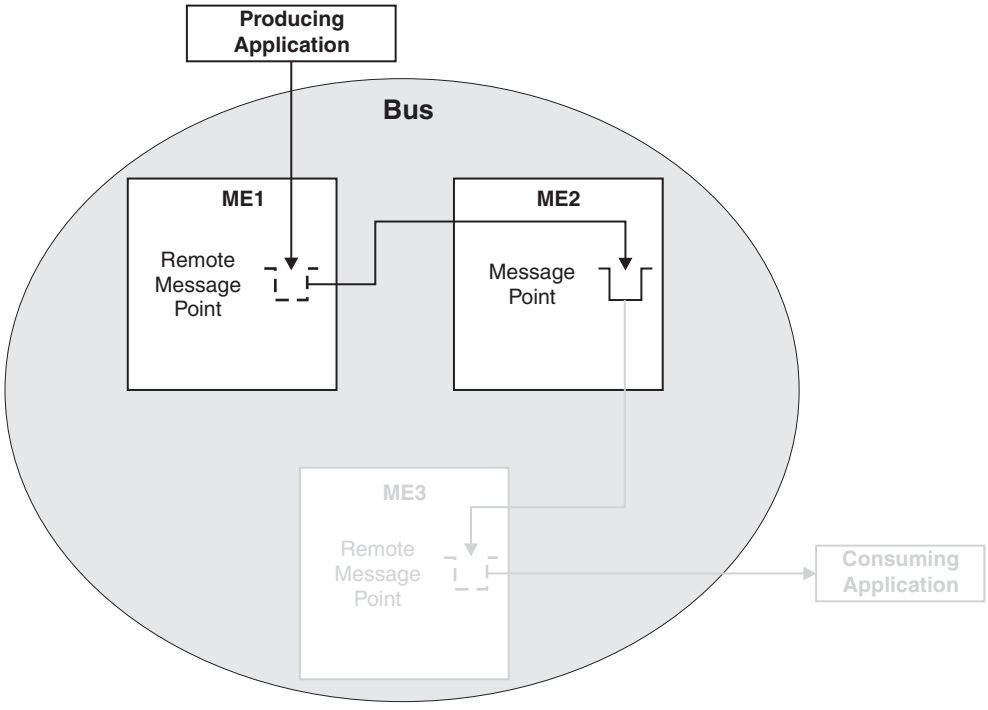


Figure 2. Point-to-point message production using a remote message point

1. Display the properties for ME1 by clicking **Service integration** → **Buses -> *bus_name* ->** [Topology] **Messaging engines -> *messaging_engine_name***.
2. On the **Runtime** tab for ME1, click [Remote message points]**Remote queue points**, then click the remote queue point that represents the queue point on ME2. Review the value of the **Current outbound messages** field.
3. If the number of current outbound messages is greater than zero, messages have been produced but they might not have been received by ME2.
   a. Check that the two messaging engines can communicate with each other, see “Service integration troubleshooting: Checking the communication between two messaging engines in a bus” on page 256.
   b. Look for previous messages on the queue. If there are previous messages, and some or all of them are for ME2, wait a few moments and then refresh the view.
      - If some of the messages have disappeared from the queue, the system is currently delivering messages but is backlogged. Wait until the backlog has been cleared, then inspect the queue point on ME2 to see if the test message has arrived.
      - If none of the messages have disappeared from the queue, the transmission of messages might be blocked by a message that is trapped in the ’Committing’ state. Later messages must wait for this message to be delivered, otherwise the ordering of messages will be broken.

        If a message is trapped in the ’Committing’ state, that message is contained in an unresolved transaction. A resource manager, such as a database, might have hung. Resolve the issue with the resource manager. If this fails, note the **Transaction ID** of the message and click **Servers -> Application servers ->** [Content pane] ***server_name* ->** [Runtime tab] [Additional Properties] **Transaction Service** to display the general properties for the transaction service. Use the **Review** links to resolve the transaction whose **Global ID** matches the transaction ID of the message.
   c. Examine the state of the test message:
      - If the status of the test message is ’Pending send’, the message is waiting to be sent. ME2 might not be accepting messages. Perform the following checks:
        – Check that the two messaging engines can communicate with each other, see “Service integration troubleshooting: Checking the communication between two messaging engines in a bus” on page 256.
        – Check that the queue point on ME2 is not full: display the runtime properties for the queue point and compare the **Current message depth** to the **High message threshold**. If the current message depth is equal to the high message threshold, the messaging engine will not accept new messages until the queued messages have been consumed. Either restart the consumer and wait until the backlog is cleared, or delete the messages.

          Note: You will not be able to recover the messages once they have been deleted.
        – Check that configuration changes have been propagated. Ensure that ME2 is aware of the existence of the queue point by deploying the latest configuration settings to ME2’s application server.
      - If the status of the test message is ’Pending acknowledgement’, the message has been sent but ME2 has either not received the message, or not processed the message. Check that there are no messages in the ’Committing’ state ahead of the test message in the transmit queue, then wait a few moments and examine the queue point again to see if the test message has arrived.If there are messages that are trapped in the ’Committing’ state, resolve this problem by referring to the following point.
      - If the test message (or another message) is in the ’Committing’ state, the message is contained in an unresolved transaction. A resource manager, such as a database, might have hung. Resolve the issue with the resource manager. If this fails, note the **Transaction ID** of the message and click **Servers -> Application servers ->** [Content pane] ***server_name* ->** [Runtime

tab] [Additional Properties] **Transaction Service** to display the general properties for the transaction service. Use the **Review** links to resolve the transaction whose **Global ID** matches the transaction ID of the message.

4. If the number of completed outbound messages is greater than zero, messages have been produced and processed by ME2, but the test message has not appeared. Rerun the producing application and ensure that the number of completed outbound messages on ME1 increases (you might see the active outbound message count increase before the completed outbound message count increases).
   - If the counts do not increase, the message was not produced at ME1. Check that the producing application is actually connected to this messaging engine (see "Determining which messaging engine an application is connected to" on page 256).
   - If the counts do increase, the message arrived at ME2, but was either consumed, sent to the exception destination, or expired. Check for the presence of consumers, and perform the preliminary checks again.
5. If the number of current and completed messages are both zero, check that the producing application really is producing messages to this destination, by performing the relevant preliminary checks again.

If you are still having problems, contact your IBM customer service representative.

Determining which messaging engine an application is connected to:

If your application fails to receive or produce a message, you might want to find out which messaging engine it is connected to, as part of troubleshooting the problem.

1. If your application is a JMS application, examine its connection factory as the messaging engine name might be specified there.
2. If your application is not a JMS application, or its connection factory does not specify the messaging engine name, use one of the following methods to determine which messaging engine the application is connected to:
   - Within the application code, after the application has obtained a valid Connection object, add a call to the toString() method of that object. The connected messaging engine name will be clearly listed when you rerun the application.
   - Enable the SIBJms_External trace component and rerun the application. Inspect the generated trace for a reference to the connected messaging engine name.

Be aware that the messaging engine name returned by either of these methods relates to the rerun of the application. It is possible that the original failing instance of the application was connected to a different messaging engine in the bus.

Service integration troubleshooting: Checking the communication between two messaging engines in a bus:

If you are troubleshooting a problem with your service integration system, you might want to check that two messaging engines can communicate with each other.

1. Check that both messaging engines are running.
2. For each messaging engine:
   a. Check the system log for a CWSIT0028I message that relates to the two messaging engines in question. This message indicates that the two messaging engines are successfully communicating with each other.
   b. Find the most recent instance (there may be only one) of the CWSIT0028I message for the two messaging engines, and check the log to make sure that a CWSIT0029I message for these two messaging engines does not appear later in the log. This message indicates that the communication connection between the two messaging engines has failed.

   If either of these checks fails, inspect the log for indications of the cause of the failure, and follow the suggested actions to rectify the problem.

Investigating why point-to-point messages are not being consumed

This topic describes how to investigate why point-to-point messages are not being consumed from a destination on a service integration bus.

Use this topic if you did not get a response in your application because a message you were expecting did not appear on a queue. The information in this topic applies to local and remote producers, and local and remote consumers.

Perform the following preliminary checks before starting the investigation:

- Perform the following preliminary checks before starting the investigation:
- Check that the consuming application is consuming messages correctly:
  - Check that the application is started.
  - Check that the name of the destination being consumed from is correct.
- Check the producing application to see if it is giving the messages a short expiry time. If this is the case, the messages may be expiring before they can be consumed.
- Click **Service integration -> Buses -> *bus_name* ->** [Destination resources] **Destinations** to display the destinations on the relevant bus. Click on the destination and check that the **Receive allowed** check box is selected.
- Check the reliability of the messages. If the reliability is set to best effort, the messages can be discarded by the system during normal operation. See “Understanding why best effort messages are being discarded” on page 255 for a list of possible causes.
- Examine the error logs.

1. Run the consuming application and check that messages are still not being consumed.
2. Stop the consuming application.
3. Determine which messaging engine is hosting the queue point to which messages are being produced. See “Determining the location of message points for a destination on a service integration bus” on page 259.
4. Click **Application servers** -> ***server_name*** -> **Messaging engines -> *messaging_engine_name* ->** [Message points] **Queue points -> *queue_point_identifier* ->** [Runtime tab] **Messages** to view the messages on the queue point. Check that there are messages present that are in the ’Unlocked’ state.
   - If there are no messages present, then there are no messages to consume. Run the producing application to produce a test message and check the queue again. If there are still no messages present, the test message has not arrived. Use the topic “Investigating why point-to-point messages are not arriving” on page 258 to investigate the problem.
   - If there are messages present but they are not in the ’Unlocked’ state, check for other consumers that are consuming from this queue point. If there are other consumers, stop them and repeat the investigation.
5. Determine which messaging engine the consuming application is connected to. See “Determining which messaging engine an application is connected to” on page 256.
   - If the consuming application is connected to the messaging engine hosting the queue point, check the consuming application for errors, in particular check that the selector in the consuming application matches the available message.
   - If the consuming application is connected to a messaging engine other than the messaging engine hosting the queue point, the messages are being routed through a remote message point. Display the runtime properties of the messaging engine that the consuming application is connected to, then display the remote message points for that messaging engine and view the list of message requests on the relevant message point. If possible, start the consuming application and ensure that it is actively trying to consume a message (the application should be in either a ’receive with wait’ state or an ’asynchronous consumer registered’ state), then follow the instructions in “Investigating why messages are not being consumed through a remote message point or subscription point, while the application is running” on page 264. If your application cannot remain in an actively consuming state

for a significant length of time (long enough to investigate the problem), follow the steps in "Investigating why messages are not being consumed through a remote message point or subscription point, while the application is stopped" on page 267.

Determining which messaging engine an application is connected to

If your application fails to receive or produce a message, you might want to find out which messaging engine it is connected to, as part of troubleshooting the problem.

1. If your application is a JMS application, examine its connection factory as the messaging engine name might be specified there.
2. If your application is not a JMS application, or its connection factory does not specify the messaging engine name, use one of the following methods to determine which messaging engine the application is connected to:
   - Within the application code, after the application has obtained a valid Connection object, add a call to the toString() method of that object. The connected messaging engine name will be clearly listed when you rerun the application.
   - Enable the SIBJms_External trace component and rerun the application. Inspect the generated trace for a reference to the connected messaging engine name.

Be aware that the messaging engine name returned by either of these methods relates to the rerun of the application. It is possible that the original failing instance of the application was connected to a different messaging engine in the bus.

Investigating why messages are not being consumed through a remote message point or subscription point, while the application is running

This topic describes how to investigate why messages are not being consumed at a destination on a service integration bus, when the messages are being routed through a remote message point and the consuming application is running.

Follow the steps in either "Investigating why point-to-point messages are not being consumed" on page 263 or "Investigating why publish/subscribe messages are not arriving at a subscription" on page 270, whichever best suits the problem. These topics contain preliminary checks and investigative tasks that you should carry out before proceeding with this task.

You should perform this task as part of either "Investigating why point-to-point messages are not being consumed" on page 263 or "Investigating why publish/subscribe messages are not arriving at a subscription" on page 270. This task explains how to investigate the flow of messages in a scenario where the messages are being routed through a remote message point and the consuming application is started. The following diagrams illustrate two possible scenarios. In Figure 1, ME2 is the messaging engine that hosts the message point, and receives messages from the producing application through ME1. ME3 is the messaging engine that the consuming application is attached to, and hosts a remote message point which represents the message point on ME2. In Figure 2, ME2 and ME3 host publication points that are represented by remote publication points on ME1, where the producing application is attached. Subscribing application B is connected to ME3 and receives messages indirectly from ME1, via a subscription on ME2. a remote subscription point on ME 3. These messaging engines are referred to in the following steps.

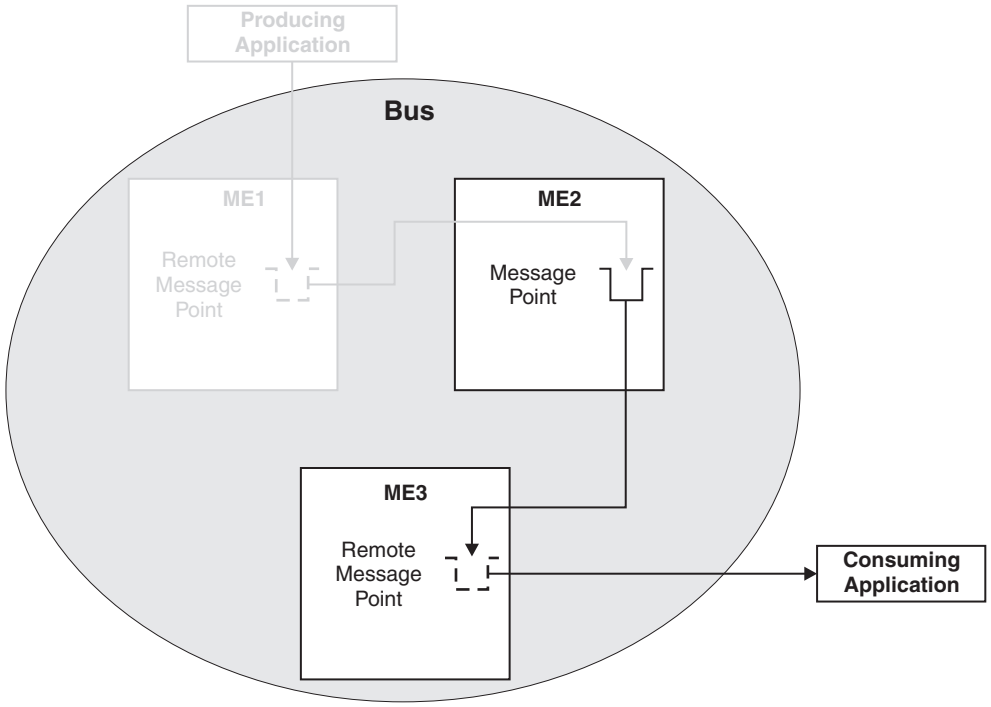


Figure 3. Point-to-point message consumption using a remote message point

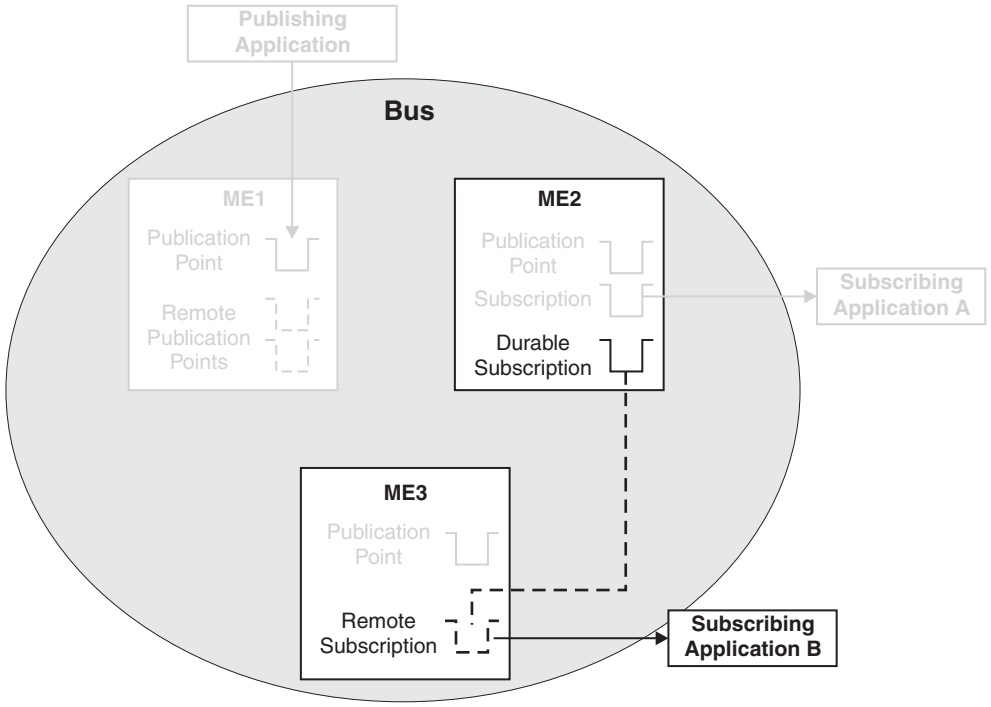


Figure 4. Publish/subscribe messaging using a remote message point

1. If you have followed the steps in “Investigating why point-to-point messages are not being consumed” on page 263 or “Investigating why publish/subscribe messages are not arriving at a subscription” on page 270 before starting this task, you should have displayed a list of message requests. Check that the list contains a request with a selector that matches an available message on the message point on ME2. If there is no such request in the list, the consuming application is not consuming; check the consuming application for errors:
   - Check that the consumer is started.
   - Check that the application is actively trying to consume:
     – If the application uses an asynchronous consumer, check that the asynchronous consumer is registered.
     – If the application is synchronous, check that the consumer is currently in a ’receive with wait’ state (this may require a modification to the application to extend the time that the application waits for a message).
2. Check the state of the active request:
   - If the state is ’Value’, a message was retrieved and returned to the consuming application, but the consumption of the message has not yet completed. Check that the consuming application is correctly processing any incoming messages, for example, check that the application is committing the transaction used to consume the message.
   - If the state is ’Rejected’, a message was retrieved and returned to the consuming application, which then rejected the message for some reason. Generally, this means that the consuming application rolled back the consume operation or an associated transaction.
   - If the state is ’Acknowledged’, an message was returned for the request and consumed by an application. Check that the message was received by the correct application, and was not consumed by a different application.
   - If the state is ’Request’, the message request has been sent to ME2, continue to the next check to investigate why a message has not been returned.
3. Note the **Request ID**. On ME2, display the message points for the destination, and view the message requests from ME3. Check that there is a request which matches the request ID on ME3. If there is no matching request, ME2 is not aware of the request. Check that the two messaging engines can communicate with each other, see “Service integration troubleshooting: Checking the communication between two messaging engines in a bus” on page 256.
4. Check the state of the request:
   - If the request is ’Requested’, the request has been received but no suitable message is available. Check that the request selector matches the available message on the message point.
   - If the request is ’Pending acknowledgement’, the request has successfully identified a matching message and attempted to transmit it to ME3. Check that the two messaging engines can communicate with each other, see “Service integration troubleshooting: Checking the communication between two messaging engines in a bus” on page 256.

If you are still having problems, contact your IBM customer service representative.

Service integration troubleshooting: Checking the communication between two messaging engines in a bus:

If you are troubleshooting a problem with your service integration system, you might want to check that two messaging engines can communicate with each other.

1. Check that both messaging engines are running.
2. For each messaging engine:
   a. Check the system log for a CWSIT0028I message that relates to the two messaging engines in question. This message indicates that the two messaging engines are successfully communicating with each other.

b. Find the most recent instance (there may be only one) of the CWSIT0028I message for the two messaging engines, and check the log to make sure that a CWSIT0029I message for these two messaging engines does not appear later in the log. This message indicates that the communication connection between the two messaging engines has failed.

If either of these checks fails, inspect the log for indications of the cause of the failure, and follow the suggested actions to rectify the problem.

Investigating why messages are not being consumed through a remote message point or subscription point, while the application is stopped

This topic describes how to investigate why messages are not being consumed at a destination on a service integration bus, when the messages are being routed through a remote message point and the consuming application is stopped.

Follow the steps in either "Investigating why point-to-point messages are not being consumed" on page 263 or "Investigating why publish/subscribe messages are not arriving at a subscription" on page 270, whichever best suits the problem. These topics contain preliminary checks and investigative tasks that you should carry out before proceeding with this task.

Perform this task as part of either "Investigating why point-to-point messages are not being consumed" on page 263 or "Investigating why publish/subscribe messages are not arriving at a subscription" on page 270. This task explains how to investigate the flow of messages in a scenario where the messages are being routed through a remote message point and the consuming application is stopped. The following diagrams illustrate two possible scenarios. In Figure 1, ME2 is the messaging engine that hosts the message point, and receives messages from the producing application through ME1. ME3 is the messaging engine that the consuming application is attached to, and hosts a remote message point which represents the message point on ME2. In Figure 2, ME2 and ME3 host publication points that are represented by remote publication points on ME1, where the producing application is attached. Subscribing application B is connected to ME3 and receives messages indirectly from ME1, via a subscription on ME2. a remote subscription point on ME 3. These messaging engines are referred to in the following steps.

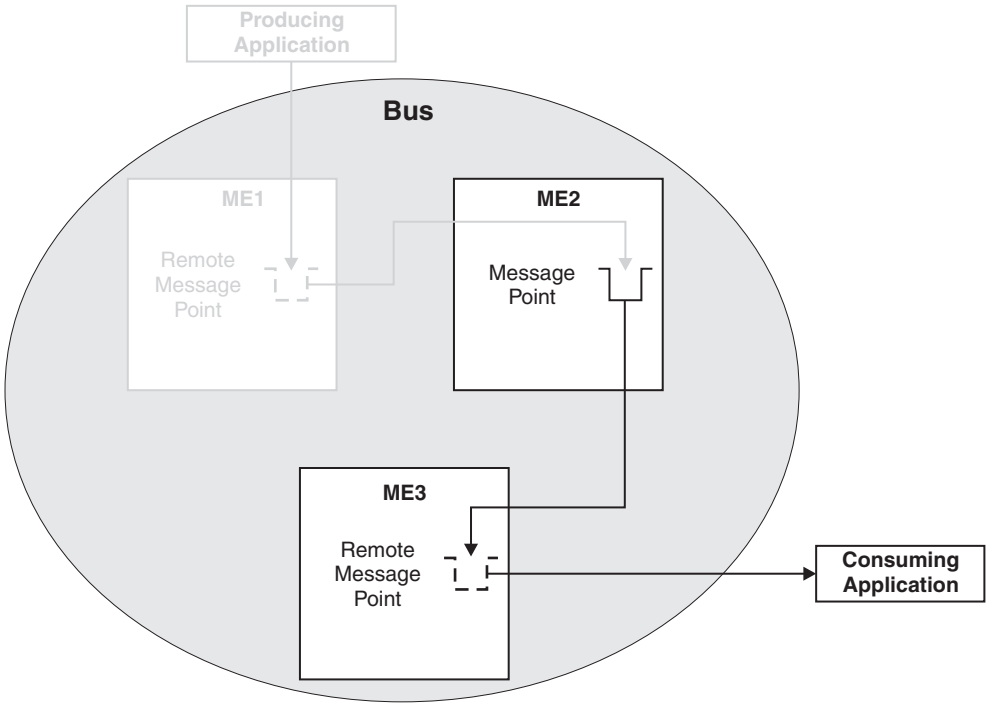


Figure 5. Point-to-point message consumption using a remote message point

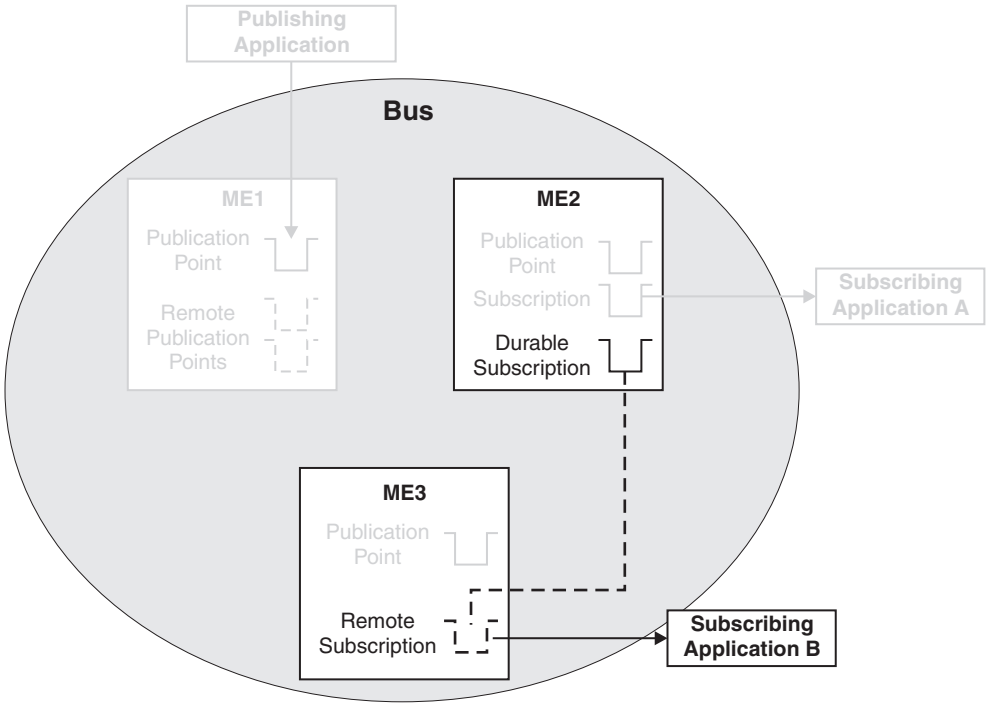


Figure 6. Publish/subscribe messaging using a remote message point

1. If you have followed the steps in “Investigating why point-to-point messages are not being consumed” on page 263 or “Investigating why publish/subscribe messages are not arriving at a subscription” on page 270 before starting this task, you should have displayed a list of message requests. On the previous panel (runtime properties for the message point), check that the **Message requests issued** (point-to-point only) or **Message requests received** (publish/subscribe only) value is greater than zero. If the value is not greater than zero, no requests have been made. Check the consuming application for errors:
   - Check that the application is actually connected to ME2.
   - Check that the application did not produce any errors which could explain why messages are not being consumed.
   - Check that the consumer was started.
   - Check that the application did attempt to consume a message:
     – If the application uses an asynchronous consumer, check that the asynchronous consumer was registered.
     – If the application is synchronous, check that the consumer performed a ’receive’ or a ’receive with wait’ function (this may require a modification to the application to extend the time that the application waits for a message).
2. If the number of issued message requests is greater than zero, requests from ME3 to ME2 for messages on the message point have been made. Check that the **Completed message requests** value is greater than zero. If not, check that the two messaging engines can communicate with each other, see “Service integration troubleshooting: Checking the communication between two messaging engines in a bus” on page 256.
3. If the number of completed message requests is greater than zero, requests are being issued by ME3, processed by ME2 and completed back to ME3. To ensure that those requests were made by the actual application being investigated, record the current values of **Completed message requests** and either **Message requests issued** or **Message requests received**. Rerun the consuming application and check that both values have increased. If the values do not increase, the application did not make a request from ME3 to ME2 for this message point (the existing numbers relate to a previous application that was consuming messages). Check the consuming application for errors:
   - Check that the application was started.
   - Check that the name of the destination being consumed from is correct.
4. If the values do increase, the message request was issued and completed, but no message was returned or processed by the consuming application.
   - Check that the application’s selection criteria match the available message or messages on the message point.
   - Check that the application is correctly receiving the message, by checking for application or runtime errors.

If you are still having problems, contact your IBM customer service representative.

Service integration troubleshooting: Checking the communication between two messaging engines in a bus:

If you are troubleshooting a problem with your service integration system, you might want to check that two messaging engines can communicate with each other.

1. Check that both messaging engines are running.
2. For each messaging engine:
   a. Check the system log for a CWSIT0028I message that relates to the two messaging engines in question. This message indicates that the two messaging engines are successfully communicating with each other.
   b. Find the most recent instance (there may be only one) of the CWSIT0028I message for the two messaging engines, and check the log to make sure that a CWSIT0029I message for these two

messaging engines does not appear later in the log. This message indicates that the communication connection between the two messaging engines has failed.

If either of these checks fails, inspect the log for indications of the cause of the failure, and follow the suggested actions to rectify the problem.

Investigating why publish/subscribe messages are not arriving at a subscription

This topic describes how to investigate why publish/subscribe messages are not arriving at a subscription on a service integration bus.

Use this topic if you have an application that is producing messages to a topic space destination, and a consuming application is not receiving the messages.

Perform the following preliminary checks before starting the investigation:

- Check that the producing application is producing messages correctly:
  - Check that there are no failures or runtime errors from the application.
  - Check that the name of the destination is correct.
  - Check that messages are actually being produced.
  - Check that the transaction used to produce the message was committed without any exceptions.
- Check that the consuming application is consuming messages correctly:
  - Check that the application is started.
  - Check that the subscription's topic and selector are correct. To do this click **Service integration -> Buses -> *bus_name* ->** [Destination resources] **Destinations -> *topic_space_name* ->** [Message points] **Publication points -> *publication_point_identifier* -> Subscriptions -> *subscription_name*** and ensure that the **Topic** and **Selector** fields match the topic and selector specified in the application.
  - If security is enabled, ensure that the subscription has the authority to receive messages sent to it. Refer to Topic security and Messaging security for more information.
- Check the producing application to see if it is giving the messages a short expiry time. If this is the case, the messages may be disappearing before they arrive, or before they can be processed by the receiving messaging engine.
- Click **Service integration -> Buses -> *bus_name* ->** [Destination resources] **Destinations** to display the destinations on the relevant bus. Click on the topic space and check that the **Send allowed** and **Receive allowed** check boxes are selected.
- Examine the relevant exception destination to see if the messages appear there. If they do, use the information contained within the messages to understand why they have arrived at the exception destination, and write an application (or mediation) to process the messages.
- Check the reliability of the messages. If the reliability is set to best effort, the messages can be discarded by the system during normal operation. See "Understanding why best effort messages are being discarded" on page 255 for a list of possible causes.
- Examine the error logs for exceptions.

1. Click **Service integration -> Buses -> *bus_name* ->** [Destination resources] **Destinations** to display the destinations on the relevant bus. Click on the relevant topic space and under **Message points**, click **Publication points**. For each publication point listed, click the publication point then click [Runtime] **Subscriptions** and look for your subscription. If your subscription is not listed on any of the publication points, there is an error in the consuming application.
2. Determine which messaging engines the producing and consuming applications are connected to, see "Determining which messaging engine an application is connected to" on page 256.

3. If the producing application is connected to the same messaging engine as the consuming application, the messages are being produced locally to the consumer. Recheck the producing and consuming applications, and check the system logs for errors.
4. If the producing application is connected to a different messaging engine than the consuming application, the messages are being routed through a remote publication point. Refer to “Investigating why publish/subscribe messages are not being received by a subscription through a remote message point” to investigate this scenario.

Determining which messaging engine an application is connected to

If your application fails to receive or produce a message, you might want to find out which messaging engine it is connected to, as part of troubleshooting the problem.

1. If your application is a JMS application, examine its connection factory as the messaging engine name might be specified there.
2. If your application is not a JMS application, or its connection factory does not specify the messaging engine name, use one of the following methods to determine which messaging engine the application is connected to:
   - Within the application code, after the application has obtained a valid Connection object, add a call to the toString() method of that object. The connected messaging engine name will be clearly listed when you rerun the application.
   - Enable the SIBJms_External trace component and rerun the application. Inspect the generated trace for a reference to the connected messaging engine name.

Be aware that the messaging engine name returned by either of these methods relates to the rerun of the application. It is possible that the original failing instance of the application was connected to a different messaging engine in the bus.

Investigating why publish/subscribe messages are not being received by a subscription through a remote message point

This topic describes how to investigate why publish/subscribe messages are not being received by a subscription on a service integration bus, when the messages are being routed through a remote message point.

Follow the steps in “Investigating why publish/subscribe messages are not arriving at a subscription” on page 270, which contains preliminary checks and investigative tasks that you should carry out before proceeding with this task.

Perform this task as part of “Investigating why publish/subscribe messages are not arriving at a subscription” on page 270. This task explains how to investigate the flow of messages in a publish/subscribe messaging scenario where the messages are being routed through a remote message point to a nondurable subscription. The following diagrams illustrates two possible situations. The dotted lines in the diagrams indicate relationships between publication points, whereas the solid lines indicate flow of messages. In Figure 1, ME1 is the messaging engine that the producing application is attached to, and ME2 and ME3 are the messaging engines that are hosting the subscriptions. The publication points on ME2 and ME3 are represented by remote publication points on ME1. In Figure 2, Subscribing application B, which is attached to ME3, has a durable subscription which receives messages through ME2, rather than directly from ME1. In order to do this, ME3 hosts a remote subscription, which represents the subscription on ME2. The messaging engines are referred to in the following steps.

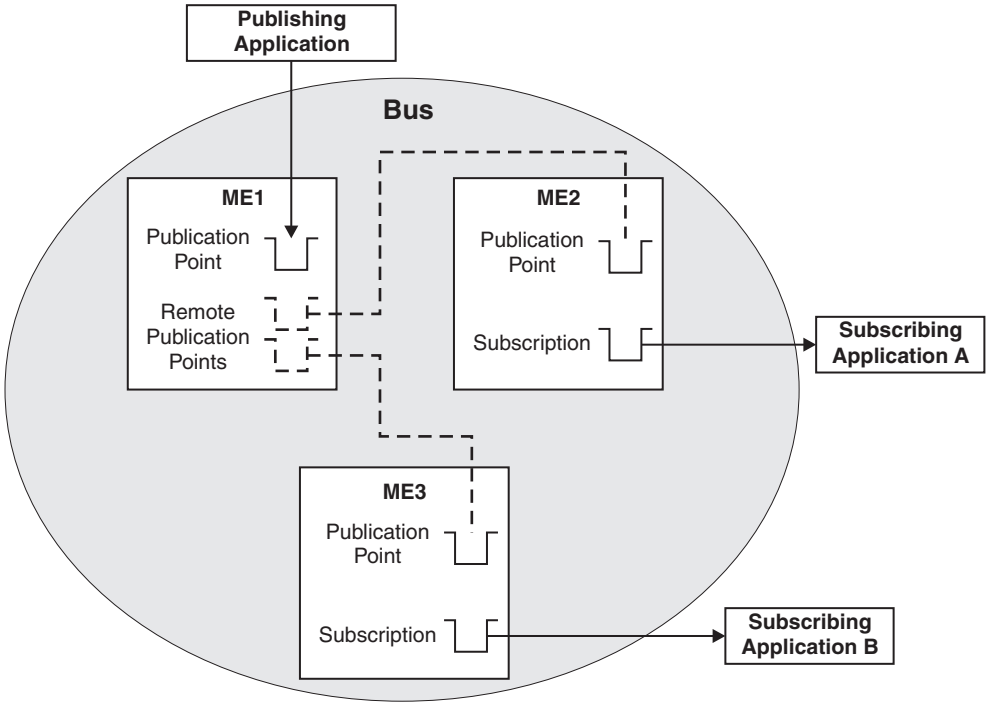


Figure 7. Point-to-point message production using a remote message point

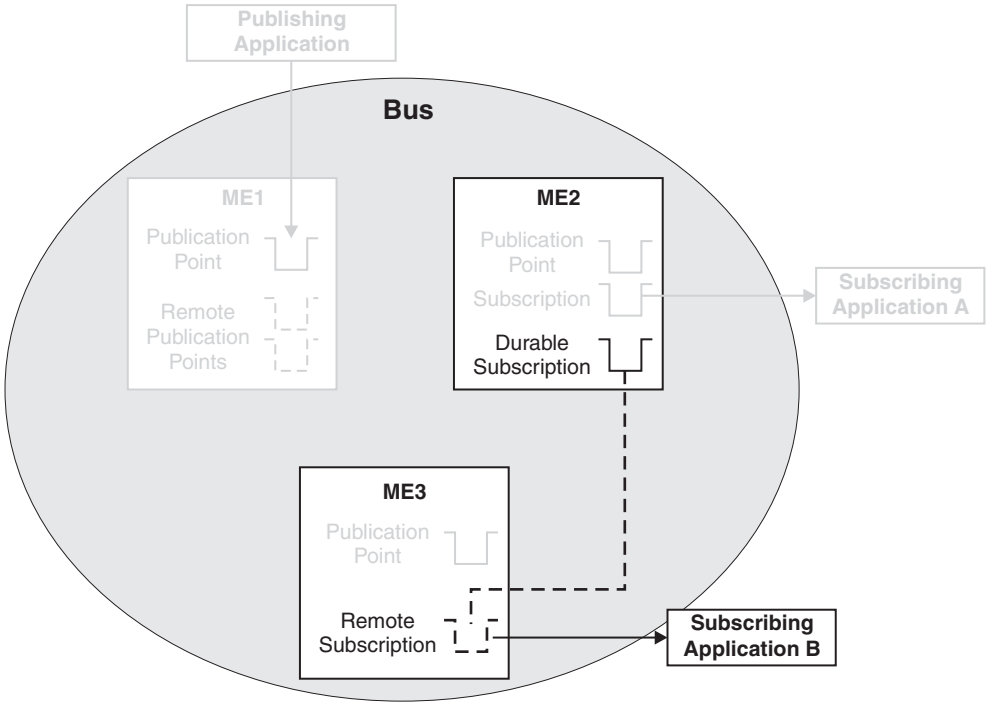


Figure 8. Publish/subscribe messaging using a remote message point

The following steps apply to both the above scenarios.

1. Display the properties for ME1 by clicking **Service integration** → **Buses ->** ***bus_name*** **->** [Topology] **Messaging engines ->** ***messaging_engine_name***.
2. On the **Runtime** tab for ME1, click [Remote message points]**Remote publication points**, then click the remote publication point that represents the publication point on ME2. Click **Topics** and check that the consumer's topic is listed. If the topic is not listed, perform the following checks:
   - Check that the subscribing application is still running.
   - Check that the two messaging engines can communicate with each other, see "Service integration troubleshooting: Checking the communication between two messaging engines in a bus" on page 256.
   - Wait a short period (dependent on the system configuration) and recheck.
3. It is possible that the registration of the subscription occurred after the message was published. Republish the message and check again to see if it was received.
4. On ME1, again display the remote publication point that represents the publication point on ME2. Review the value of the **Current outbound messages** field.
5. If the number of current outbound messages is greater than zero, messages have been produced but they might not have been received by ME2.
   a. Check that the two messaging engines can communicate with each other, see "Service integration troubleshooting: Checking the communication between two messaging engines in a bus" on page 256.
   b. Look for previous messages on the topic space. If there are previous messages, and some or all of them are for ME2, wait a few moments and then refresh the view.
      - If some of the messages have disappeared from the topic space, the system is currently delivering messages but is backlogged. Wait until the backlog has been cleared, then inspect the publication point on ME2 to see if the test message has arrived.
      - If none of the messages have disappeared from the topic space, the transmission of messages might be blocked by a message that is trapped in the 'Committing' state. Later messages must wait for this message to be delivered, otherwise the ordering of messages will be broken.

        If a message is trapped in the 'Committing' state, that message is contained in an unresolved transaction. A resource manager, such as a database, might have hung. Resolve the issue with the resource manager. If this fails, note the **Transaction ID** of the message and click **Servers -> Application servers ->** [Content pane] ***server_name*** **->** [Runtime tab] [Additional Properties] **Transaction Service** to display the general properties for the transaction service. Use the **Review** links to resolve the transaction whose **Global ID** matches the transaction ID of the message.
   c. Examine the state of the test message:
      - If the status of the test message is 'Pending send', the message is waiting to be sent. ME2 might not be accepting messages. Perform the following checks:
        – Check that the two messaging engines can communicate with each other, see "Service integration troubleshooting: Checking the communication between two messaging engines in a bus" on page 256.
        – Check that the publication point on ME2 is not full: display the runtime properties for the publication point and compare the **Current message depth** to the **High message threshold**. If the current message depth is equal to the high message threshold, the messaging engine will not accept new messages until the queued messages have been consumed. Either restart the consumer and wait until the backlog is cleared, or delete the messages.
        – Check that configuration changes have been propagated. Ensure that ME2 is aware of the existence of the publication point by deploying the latest configuration settings to ME2's application server.
      - If the status of the test message is 'Pending acknowledgement', the message has been sent but ME2 has either not received the message, or not processed the message. Check that there

are no messages in the 'Committing' state ahead of the test message in the transmit queue, then wait a few moments and examine the publication point again to see if the test message has arrived.If there are messages that are trapped in the 'Committing' state, resolve this problem by referring to the following point.

- If the test message (or another message) is in the 'Committing' state, the message is contained in an unresolved transaction. A resource manager, such as a database, might have hung. Resolve the issue with the resource manager. If this fails, note the **Transaction ID** of the message and click **Servers -> Application servers ->** [Content pane] ***server_name* ->** [Runtime tab] [Additional Properties] **Transaction Service** to display the general properties for the transaction service. Use the **Review** links to resolve the transaction whose **Global ID** matches the transaction ID of the message.

6. If the number of completed outbound messages is greater than zero, messages have been produced and processed by ME2, but the test message has not appeared. Rerun the producing application and ensure that the number of completed outbound messages on ME1 increases (you might see the active outbound message count increase before the completed outbound message count increases).
   - If the counts do not increase, the message was not produced at ME1. Check that the producing application is actually connected to this messaging engine (see "Determining which messaging engine an application is connected to" on page 256).
   - If the counts do increase, the message arrived at ME2, but was either consumed, sent to the exception destination, or expired. Check for the presence of consumers, and perform the preliminary checks again.
7. If the number of current and completed messages are both zero, check that the producing application really is producing messages to this destination, by performing the relevant preliminary checks again.
8. You have now checked the flow of messages between ME1 and ME2. If you have an application which is in the position of Subscribing application A or B in Figure 1, you have investigated the situation fully; if you are still having problems, contact your IBM customer service representative. If you have an application which is in the position of Subscribing application B in Figure 2, in other words an application which has a remote subscription, you also need to investigate the flow of messages between ME2 and ME3, using the following steps. To determine if your application is using a remote subscription, display the publication points for the relevant topic space, find your subscription and examine the name of the publication point. The name will be of the form *topic_space_name@messaging_engine_name*. This will tell you which messaging engine the subscription is hosted by. If this messaging engine is different than both the messaging engine that the producing application is connected to, and the messaging engine that the consuming application is connected to, a remote subscription is being used.
9. Display the subscriptions for the publication point on ME2, and find your subscription in the list. If the subscription is not listed, perform the following checks:
   - Check that the subscribing application is still running.
   - Check that the two messaging engines can communicate with each other, see "Service integration troubleshooting: Checking the communication between two messaging engines in a bus" on page 256.
   - Wait a short period (dependent on the system configuration) and recheck.
10. Click your subscription and then click **Known remote subscription points**. In the resulting list, click the name of the messaging engine that is represented by ME3 in the diagram. Click **Message requests**. This displays the requests that have been received by the subscription on ME2, from the remote subscription on ME3.
11. If possible, start the consuming application and ensure that it is actively trying to consume a message (the application should be in either a 'receive with wait' state or an 'asynchronous consumer registered' state), then follow the instructions in "Investigating why messages are not being consumed through a remote message point or subscription point, while the application is running" on page 264. If your application cannot remain in an actively consuming state for a significant length of time (long

enough to investigate the problem), follow the steps in “Investigating why messages are not being consumed through a remote message point or subscription point, while the application is stopped” on page 267.

Service integration troubleshooting: Checking the communication between two messaging engines in a bus:

If you are troubleshooting a problem with your service integration system, you might want to check that two messaging engines can communicate with each other.

1. Check that both messaging engines are running.
2. For each messaging engine:
   a. Check the system log for a CWSIT0028I message that relates to the two messaging engines in question. This message indicates that the two messaging engines are successfully communicating with each other.
   b. Find the most recent instance (there may be only one) of the CWSIT0028I message for the two messaging engines, and check the log to make sure that a CWSIT0029I message for these two messaging engines does not appear later in the log. This message indicates that the communication connection between the two messaging engines has failed.

   If either of these checks fails, inspect the log for indications of the cause of the failure, and follow the suggested actions to rectify the problem.

Determining which messaging engine an application is connected to:

If your application fails to receive or produce a message, you might want to find out which messaging engine it is connected to, as part of troubleshooting the problem.

1. If your application is a JMS application, examine its connection factory as the messaging engine name might be specified there.
2. If your application is not a JMS application, or its connection factory does not specify the messaging engine name, use one of the following methods to determine which messaging engine the application is connected to:
   - Within the application code, after the application has obtained a valid Connection object, add a call to the toString() method of that object. The connected messaging engine name will be clearly listed when you rerun the application.
   - Enable the SIBJms_External trace component and rerun the application. Inspect the generated trace for a reference to the connected messaging engine name.

Be aware that the messaging engine name returned by either of these methods relates to the rerun of the application. It is possible that the original failing instance of the application was connected to a different messaging engine in the bus.

Chapter 20. Data access resources

Connection and connection pool statistics

Performance Monitoring Infrastructure (PMI) method calls that are supported in the two existing Connection Managers (JDBC and J2C) are still supported in this version of WebSphere Application Server. The calls include:

- ManagedConnectionsCreated
- ManagedConnectionsAllocated
- ManagedConnectionFreed
- ManagedConnectionDestroyed
- BeginWaitForConnection
- EndWaitForConnection
- ConnectionFaults
- Average number of ManagedConnections in the pool
- Percentage of the time that the connection pool is using the maximum number of ManagedConnections
- Average number of threads waiting for a ManagedConnection
- Average percent of the pool that is in use
- Average time spent waiting on a request
- Number of ManagedConnections that are in use
- Number of Connection Handles
- FreePoolSize
- UseTime

Java Specification Request (JSR) 77 requires statistical data to be accessed through managed beans (Mbeans) to facilitate this. The Connection Manager passes the ObjectNames of the Mbeans created for this pool. In the case of Java Message Service (JMS) *null* is passed in. The interface used is:

```
PmiFactory.createJ2CPerf(
    String pmiName, // a unique Identifier for JCA /JDBC.  This is the
                    // ConnectionFactory name.

    ObjectName providerName,// the ObjectName of the J2CResourceAdapter
                             // or JDBCProvider Mbean

    ObjectName factoryName // the ObjectName of the J2CConnectionFactory
                           // or DataSourceMbean.
)
```

The following Unified Modeling Language (UML) diagram shows how JSR 77 requires statistics to be reported:

In WebSphere Application Server Version 5.x, the JCAStats interface was implemented by the J2CResourceAdapter Mbean, and the JDBCStats interface was implemented by the JDBCProvider Mbean. The JCAConnectionStats and JDBCConnectionStats interfaces are not implemented because they collect statistics for nonpooled connections, which are not present in the JCA 1.0 Specification. JCAConnectionPoolStats, and JDBCConnectionPoolStats do not have a direct implementing Mbean; those statistics are gathered through a call to PMI. A J2C resource adapter, and JDBC provider each contain a list of ConnectionFactory or DataSource ObjectNames, respectively. The ObjectNames are used by PMI to find the appropriate connection pool in the list of PMI modules.

The JCA 1.5 Specification allows an exception from the matchManagedConnection() method that indicates that the resource adapter requests that the connection not be pooled. In that case, statistics for that connection are provided separately from the statistics for the connection pool.

Example: Connection factory lookup

```
import javax.resource.cci.*;
import javax.resource.ResourceException;

import javax.naming.*;

import java.util.*;

/**
 * This class is used to look up a connection factory.
 */
public class ConnectionFactoryLookup {

   String jndiName = "java:comp/env/eis/SampleConnection";
   boolean verbose = false;

   /**
    * main method
    */
```

```
  public static void main(String[] args) {
     ConnectionFactoryLookup cfl = new ConnectionFactoryLookup();
     cfl.checkParam(args);

     try {
        cfl.lookupConnectionFactory();
     }
     catch(javax.naming.NamingException ne) {
  System.out.println("Caught this " + ne);
  ne.printStackTrace(System.out);
     }
     catch(javax.resource.ResourceException re) {
  System.out.println("Caught this " + re);
  re.printStackTrace(System.out);
     }
  }

/**
 * This method does a simple Connection Factory lookup.
   *
   * After the Connection Factory is looked up, a connection is got from
 * the Connection Factory.  Then the Connection MetaData is retrieved
   * to verfiy the connection is workable.
 */
public void lookupConnectionFactory()
 throws javax.naming.NamingException, javax.resource.ResourceException {

 javax.resource.cci.ConnectionFactory factory = null;
 javax.resource.cci.Connection conn = null;
 javax.resource.cci.ConnectionMetaData metaData = null;

 try {
  // lookup the connection factory
        if (verbose) System.out.println("Look up the connection factory...");

  InitialContext ic = new InitialContext();
  factory = (ConnectionFactory) ic.lookup(jndiName);

  // Get connection
        if (verbose) System.out.println("Get the connection...");
  conn = factory.getConnection();

  // Get ConnectionMetaData
  metaData = conn.getMetaData();

  // Print out the metadata Informatin.
  if (verbose) System.out.println("  **  EISProductName   :" + metaData.getEISProductName());
  if (verbose) System.out.println("      EISProductVersion:" + metaData.getEISProductName());
  if (verbose) System.out.println("      UserName         :" + metaData.getUserName());

        System.out.println("Connection factory "+jndiName+" is successfully looked up");
 }
 catch (javax.naming.NamingException ne) {
  // Connection factory cannot be looked up.
  throw ne;
 }
 catch (javax.resource.ResourceException re) {
  // Something wrong with connections.
  throw re;
 }
 finally {
  if (conn != null) {
   try {
    conn.close();
   }
   catch (javax.resource.ResourceException re) {
   }
```

```
   }
  }
 }

   /**
    * Check and gather all the parameters.
    */
   private void checkParam(String args[]) {
  int i = 0, j;
  String arg;
  char flag;
      boolean help = false;

  // parse out the options
  while (i < args.length && args[i].startsWith("-")) {
   arg = args[i++];

   // get the database name
   if (arg.equalsIgnoreCase("-jndiName")) {
    if (i < args.length)
     jndiName = args[i++];
    else {
     System.err.println("-jndiName requires a J2C Connection Factory JNDI name");
     break;
    }
   }
   else { // check for verbose, cmp , bmp
    for (j = 1; j < arg.length(); j++) {
     flag = arg.charAt(j);
     switch (flag) {
      case 'v' :
      case 'V' :
       verbose = true;
       break;
                 case 'h' :
                 case 'H' :
                     help = true;
                     break;
      default :
       System.err.println("illegal option " + flag);
       break;
     }
    }
   }
  }

  if ((i != args.length) || help) {
   System.err.println("Usage: java ConnectionFactoryLookup [-v] [-h]");
   System.err.println("    [-jndiName the J2C Connection Factory JNDI name]");
   System.err.println("-v=verbose");
   System.err.println("-h=this information");
   System.exit(1);
  }
   }

}
```

Vendor-specific data sources minimum required settings

Use this table as an at-a-glance reference of JDBC providers that can be defined for use with WebSphere Application Server Version 6.x, to establish data sources for transacting with relational databases. A list that contains detailed requirements for creating data sources with these providers follows the table. (The list also contains information about JDBC providers that are *deprecated* in WebSphere Application Server Version 6.x.)

Database type	JDBC Provider	Transaction support	Version and other considerations
DB2 on Windows, UNIX, or workstation-based LINUX	DB2 Universal JDBC Provider	One phase only	
	DB2 Universal JDBC Provider (XA)	One and two phase	The XA implementation is *not* supported in WebSphere Application Server run on workstation-based LINUX
	DB2 legacy CLI-based Type 2 JDBC Provider *Deprecated in WebSphere Application Server v.6.1*	One phase only	
	DB2 legacy CLI-based Type 2 JDBC Provider (XA) *Deprecated in WebSphere Application Server v.6.1*	One and two phase	
DB2 UDB for iSeries	DB2 UDB for iSeries (Native)	One phase only	Recommended for use with WebSphere Application Server **run on iSeries**
	DB2 UDB for iSeries (Native XA)	One and two phase	Recommended for use with WebSphere Application Server **run on iSeries**
	DB2 UDB for iSeries (Toolbox)	One phase only	
	DB2 UDB for iSeries (Toolbox XA)	One and two phase	
	DB2 Universal JDBC Provider (XA)	One and two phase	**-***Only* for use with WebSphere Application Server **run on z/OS** **-**Only driver type 4 is supported **-**Does *not* support Version 4 data sources
	DB2 legacy CLI-based Type 2 JDBC Provider *Deprecated in WebSphere Application Server v.6.1*	One phase only	**-***Only* for use with WebSphere Application Server run on Windows, UNIX, or workstation-based LINUX **-** Requires the DB2 Connect driver (available from DB2)
	DB2 legacy CLI-based Type 2 JDBC Provider (XA) *Deprecated in WebSphere Application Server v.6.1*	One and two phase	**-***Only* for use with WebSphere Application Server run on Windows, UNIX, or workstation-based LINUX **-** Requires the DB2 Connect driver (available from DB2)

Database type	JDBC Provider	Transaction support	Version and other considerations
DB2 on z/OS	DB2 for z/OS Local JDBC Provider (RRS), using the Legacy DB2 for OS/390 and z/OS JDBC Driver **Removed support:** WebSphere Application Server v6.1 does not support this provider. Use the DB2 Universal JDBC Driver provider instead. See ″Migrating from the JDBC/SQLJ Driver for OS/390 and z/OS to the DB2 Universal JDBC Driver″ in the Information Management Software for z/OS Solutions Information Center.	One and two phase	*Only* for use with WebSphere Application Server **run on z/OS**
	DB2 Universal JDBC Provider	One phase only	
	DB2 Universal JDBC Provider (XA)	One and two phase	**-**Only driver type 4 is supported in WebSphere Application Server **run on z/OS** **-**Does *not* support Version 4 data sources in WebSphere Application Server **run on z/OS**
	DB2 legacy CLI-based Type 2 JDBC Provider *Deprecated in WebSphere Application Server v.6.1*	One phase only	**-***Only* for use with WebSphere Application Server run on Windows, UNIX, or workstation-based LINUX **-** Requires the DB2 Connect program (available from DB2)
	DB2 legacy CLI-based Type 2 JDBC Provider (XA) *Deprecated in WebSphere Application Server v.6.1*	One and two phase	**-***Only* for use with WebSphere Application Server run on Windows, UNIX, or workstation-based LINUX **-** Requires the DB2 Connect program (available from DB2)

Database type	JDBC Provider	Transaction support	Version and other considerations
Cloudscape Version 10.1.x *Cloudscape v10.1.x provides the Apache Derby runtime.* **JDBC naming requirements:** Because the new Cloudscape code base is the product of the Apache Derby Project, Cloudscape v10.1.x uses Derby JDBC classes. Therefore you must specify Derby JDBC provider names, Derby JDBC driver classes, and Derby data source classes to configure JDBC access to Cloudscape v10.1.x.	Derby JDBC Provider	One phase only	- Not for use in clustered environment: accessible from a single JVM only - Does not support Version 4 data sources
	Derby JDBC Provider (XA)	One and two phase	- Not for use in clustered environment: accessible from a single JVM only - Does not support Version 4 data sources
	Derby Network Server Provider using the Universal JDBC driver *Deprecated in WebSphere Application Server v.6.1*	One phase only	- *Can be used in clustered environment:* a database instance can be accessed by multiple JVMs - Does not support XA - Does not support Version 4 data sources
	Derby Network Server using Derby Client provider	One phase only	- *Can be used in clustered environment:* a database instance can be accessed by multiple JVMs - Does not support Version 4 data sources - Only for use with Cloudscape 10.1.x databases that run on the same node as WebSphere Application Server
	Derby Network Server using Derby Client provider (XA)	One and two phase	- *Can be used in clustered environment:* a database instance can be accessed by multiple JVMs - Does not support Version 4 data sources - Only for use with Cloudscape 10.1.x databases that run on the same node as WebSphere Application Server
Informix	Informix JDBC Provider	One phase only	
	Informix JDBC Provider (XA)	One and two phase	
Sybase	Sybase jConnect for JDBC Provider	One phase only	
	Sybase jConnect for JDBC Provider (XA)	One and two phase	

Database type	JDBC Provider	Transaction support	Version and other considerations
Oracle	Oracle JDBC Provider	One phase only	
	Oracle JDBC Provider(XA)	One and two phase	
Microsoft SQL Server	DataDirect ConnectJDBC Provider, type 4 driver, for MS SQL Server	One phase only	- Only for use with the corresponding driver from DataDirect Technologies - Version 3.5, Service pack 2 of the DataDirect ConnectJDBC type 4 driver can support access to both MS SQL Server 2005 and MS SQL Server 2000
	DataDirect ConnectJDBC Provider, type 4 driver, for MS SQL Server (XA)	One and two phase	- Only for use with the corresponding driver from DataDirect Technologies - Version 3.5, Service pack 2 of the DataDirect ConnectJDBC type 4 driver can support access to both MS SQL Server 2005 and MS SQL Server 2000
	IBM WebSphere embedded ConnectJDBC Provider for MS SQL Server	One phase only	- Cannot be used outside of WebSphere Application Server environment - Version 3.5, Service pack 2 of the driver can support access to both MS SQL Server 2005 and MS SQL Server 2000
	IBM WebSphere embedded ConnectJDBC Provider for MS SQL Server (XA)	One and two phase	- Cannot be used outside of WebSphere Application Server environment - Version 3.5, Service pack 2 of the driver can support access to both MS SQL Server 2005 and MS SQL Server 2000

Detailed data source requirements per JDBC provider and platform

The following list contains the requirements for creating data sources with every JDBC provider type that is supported in WebSphere Application Server Version 6.x. Specific fields are designated for the user and password properties. Inclusion of a property in the list does not imply that you should add it to the data source custom properties list. Rather, inclusion in the list means that a value is *typically* required for that field.

Important: After you determine the type of JDBC provider that suits your application and environment, ensure that you acquire the corresponding JDBC driver at a release level supported by this version of WebSphere Application Server. Consult the WebSphere Application Server prerequisite Web site.

Use these links to find your provider and data source information:

- DB2 UDB for iSeries
- “Cloudscape v10.x” on page 288
  - Formerly known as Derby
  - Comprised of the Derby code base
  - Not supported as a production database with this version of WebSphere Application Server
  - **Cloudscape v10.1.x:** The Cloudscape 10.1.x package that is bundled with WebSphere Application Server is backed by full IBM Quality Assurance (QA).
- Informix
- Sybase
- Oracle
- MS SQL Server

DB2 UDB for iSeries

1. **DB2 UDB for iSeries (Native)**

   The iSeries Developer Kit for Java contains this Type 2 JDBC driver that is built on top of the iSeries DB2 Call Level Interface (CLI) native libraries. Only use this driver for local DB2 connections on iSeries. It is not recommended for remote access. Use this driver for iSeries V5R2, or later releases.

   DB2 UDB for iSeries (Native V5R2 and later) supports one phase data source:

   com.ibm.db2.jdbc.app.UDBConnectionPoolDataSource

   Requires JDBC driver files: **db2_classes.jar**

   Requires **DataStoreHelper** class:

   com.ibm.websphere.rsadapter.DB2AS400DataStoreHelper

   Does not require an authentication alias.

   Requires properties:
   - **databaseName** The name of the relational database to which the data source connections are established. This name must appear in the iSeries Relational Database Directory. The default is *LOCAL.

2. **DB2 UDB for iSeries (Native XA)**

   The iSeries Developer Kit for Java contains this XA-compliant Type 2 JDBC driver built on top of the iSeries DB2 Call Level Interface (CLI) native libraries. Only use this driver for local DB2 connections on iSeries. It is not recommended for remote access. Use this driver for iSeries V5R2 or later releases.

   DB2 UDB for iSeries (Native XA - V5R2 and later) supports two phase data source:

   com.ibm.db2.jdbc.app.UDBXADataSource

   Requires JDBC driver files: **db2_classes.jar**

   Requires **DataStoreHelper** class:

   com.ibm.websphere.rsadapter.DB2AS400DataStoreHelper

   Does not require an authentication alias.

   Requires properties:
   - **databaseName** The name of the relational database to which the data source connections are established. This name must appear in the iSeries Relational Database Directory. The default is *LOCAL.

3. **DB2 UDB for iSeries (Toolbox)**

   This JDBC driver, also known as iSeries Toolbox driver for Java, is provided in the DB2 for iSeries database server. Use this driver for remote DB2 connections on iSeries. We recommend you use this driver instead of the IBM Developer Kit for Java JDBC Driver to access remote DB2 UDB for iSeries systems.

   DB2 UDB for iSeries (Toolbox) supports one phase data source:

   com.ibm.as400.access.AS400JDBCConnectionPoolDataSource

   Requires JDBC driver files: **jt400.jar**

   Requires **DataStoreHelper** class:

   com.ibm.websphere.rsadapter.DB2AS400DataStoreHelper

Does not require an authentication alias if WebSphere Application Server and DB2 UDB for iSeries are installed in the same server. If they are installed in different servers, the user ID and password are required.

Requires properties:

- **serverName** The name of the server from which the data source obtains connections. Example: *myserver.mydomain.com*.

4. **DB2 UDB for iSeries (Toolbox XA)**

   This XA compliant JDBC driver, also known as iSeries Toolbox XA compliant driver for Java, is provided in the DB2 for iSeries database server. Use this driver for remote DB2 connections on iSeries. We recommend you use this driver instead of the IBM Developer Kit for Java JDBC Driver to access remote DB2 UDB for iSeries systems.

   DB2 UDB for iSeries (Toolbox XA) supports two phase data source:

   com.ibm.as400.access.AS400JDBCXADataSource

   Requires JDBC driver files: **jt400.jar**

   Requires **DataStoreHelper** class:

   com.ibm.websphere.rsadapter.DB2AS400DataStoreHelper

   Does not require an authentication alias if WebSphere Application Server and DB2 UDB for iSeries are installed in the same server. If they are installed in different servers, the user ID and password are required.

   Requires properties:

   - **serverName** The name of the server from which the data source obtains connections. Example: *myserver.mydomain.com*.

5. **DB2 legacy CLI-based Type 2 JDBC Driver**

   Deprecated in Version 6.1: This JDBC provider is deprecated in WebSphere Application Server Version 6.1. Therefore it is no longer an available choice among provider types in the administrative console. Select the DB2 Universal JDBC Provider instead.

   The DB2 legacy CLI-based Type 2 JDBC Driver Provider is built on top of DB2 CLI (Call Level Interface). It uses the DB2 CLI interface to communicate with DB2 UDB servers. This provider is intended for *remote* connections to DB2 running on iSeries; for use with Application Server on Windows, UNIX, or workstation-based LINUX, it therefore requires the DB2 Connect Driver (which is available from DB2).

   DB2 legacy CLI-based Type 2 JDBC Driver supports one phase data source:

   COM.ibm.db2.jdbc.DB2ConnectionPoolDataSource

   Requires JDBC driver files: **db2java.zip** (Note: If you run SQLJ in DB2 Version 8, **db2jcc.jar** is also required.)

   Requires **DataStoreHelper** class:

   com.ibm.websphere.rsadapter.DB2DataStoreHelper

   Does not require a valid authentication alias.

   Requires properties:

   - **databaseName** The name of the database from which the data source obtains connections. Example: *Sample*.

6. **DB2 legacy CLI-based Type 2 JDBC Driver (XA)**

   Deprecated in Version 6.1: This JDBC provider is deprecated in WebSphere Application Server Version 6.1. Therefore it is no longer an available choice among provider types in the administrative console. Select the DB2 Universal JDBC Provider (XA) instead.

   The DB2 legacy CLI-based Type 2 JDBC Driver (XA) is built on top of DB2 CLI (Call Level Interface). It uses the DB2 CLI interface to communicate with DB2 UDB servers. This provider is intended for

remote connections to DB2 running on iSeries; for use with Application Server on Windows, UNIX, or workstation-based LINUX, it therefore requires the DB2 Connect Driver (which is available from DB2).

DB2 legacy CLI-based Type 2 JDBC Driver (XA) supports two phase data source:

COM.ibm.db2.jdbc.DB2XADataSource

Requires JDBC driver files: **db2java.zip**

Requires **DataStoreHelper** class:

`com.ibm.websphere.rsadapter.DB2DataStoreHelper`

Does not require a valid authentication alias.

Requires properties:

- **databaseName** The name of the database from which the data source obtains connections. Example: *Sample*.

7. **DB2 UDB for iSeries (Native - Version 5 Release 1 and earlier)** -- Deprecated

   This JDBC provider is deprecated because it corresponds to a version of the iSeries operating system that WebSphere Application Server Version 6.x does not support. You must now use iSeries V5R2 or a later release of the iSeries operating system, for which the WebSphere Application Server administrative console lists one native iSeries DB2 non-XA provider: DB2 UDB for iSeries (Native).

   The iSeries Developer Kit for Java contains this Type 2 JDBC driver that is built on top of the iSeries DB2 Call Level Interface (CLI) native libraries. Only use this driver for local DB2 connections on iSeries. It is not recommended for remote access. Use this driver for iSeries V5R1, or earlier releases.

   DB2 UDB for iSeries (Native V5R1 and earlier) supports one phase data source:

   com.ibm.db2.jdbc.app.DB2StdConnectionPoolDataSource

   Requires JDBC driver files: **db2_classes.jar**

   Requires **DataStoreHelper** class:

   `com.ibm.websphere.rsadapter.DB2AS400DataStoreHelper`

   Does not require an authentication alias.

   Requires properties:

   - **databaseName** The name of the relational database to which the data source connections are established. This name must appear in the iSeries Relational Database Directory. The default is *LOCAL.

8. **DB2 UDB for iSeries (Native XA - Version 5 Release 1 and earlier)** -- Deprecated

   This JDBC provider is deprecated because it corresponds to a version of the iSeries operating system that WebSphere Application Server Version 6.x does not support. You must now use iSeries V5R2 or a later release of the iSeries operating system, for which the administrative console lists one native iSeries DB2 XA provider: DB2 UDB for iSeries (Native XA).

   The iSeries Developer Kit for Java contains this XA-compliant Type 2 JDBC driver built on top of the iSeries DB2 Call Level Interface (CLI) native libraries. Only use this driver for local DB2 connections on iSeries. It is not recommended for remote access. Use this driver for iSeries V5R1, or earlier releases.

   DB2 UDB for iSeries (Native XA - V5R1 and earlier) supports two phase data source:

   com.ibm.db2.jdbc.app.DB2StdXADataSource

   Requires JDBC driver files: **db2_classes.jar**

   Requires **DataStoreHelper** class:

   `com.ibm.websphere.rsadapter.DB2AS400DataStoreHelper`

   Does not require an authentication alias.

   Requires properties:

   - **databaseName** The name of the relational database to which the data source connections are established. This name must appear in the iSeries Relational Database Directory. The default is *LOCAL.

For more information on DB2 UDB for iSeries, visit the DB2 Web site at: http://www.ibm.com/software/data/db2/

Cloudscape v10.x

(Cloudscape v10.0 through Cloudscape 10.1.x)

•

Requirement: To configure JDBC access, you must specify Derby JDBC provider names, Derby JDBC driver classes, and Derby data source classes. Because the new Cloudscape code base is the product of the Apache Derby Project, the database uses Derby JDBC classes.

•

Restriction: Do not use Cloudscape v10.x as a production database. Use it for development and test purposes only.

1. **Derby JDBC Provider**

   The Derby JDBC driver provides JDBC access to the Cloudscape v10.x database by using the framework that is already embedded in WebSphere Application Server for Cloudscape. However, you cannot use any Version 4.0 data sources with Cloudscape v10.x.

   The Derby JDBC Provider supports one phase data source:

   org.apache.derby.jdbc.EmbeddedConnectionPoolDataSource

   Requires JDBC driver files: **derby.jar**; full path name: `${WAS_APP_SERVER_ROOT}/derby/lib/derby.jar`

   Requires **DataStoreHelper** class:

   `com.ibm.websphere.rsadapter.DerbyDataStoreHelper`

   Does not require a valid authentication alias.

   Requires properties:

   - **databaseName** The name of the database from which the data source obtains connections. If you do not specify a fully qualified path name, Application Server uses the default location of `WAS_HOME/derby` (or the equivalent default for a UNIX or LINUX environment).
     – Example database path name for Windows: `c:\temp\sampleDB`
     – Example database path name for UNIX or LINUX: `/tmp/sampleDB`

     If no database currently exists for the path name you want to specify, simply append `;create=true` to the path name to create a database dynamically. (For example: `c:\temp\sampleDB;create=true`)

2. **Derby JDBC Provider (XA)**

   The Derby JDBC driver (XA) provides JDBC access to the Cloudscape v10.x database by using the framework that is already embedded in WebSphere Application Server for Cloudscape. However, you cannot use any Version 4.0 data sources with Cloudscape v10.x.

   The Derby JDBC Provider (XA) supports two phase data source:

   org.apache.derby.jdbc.EmbeddedXADataSource

   Requires JDBC driver files: **derby.jar**; full path name: `${WAS_APP_SERVER_ROOT}/derby/lib/derby.jar`

   Requires **DataStoreHelper** class:

   `com.ibm.websphere.rsadapter.DerbyDataStoreHelper`

   Does not require a valid authentication alias.

   Requires properties:

   - **databaseName** The name of the database from which the data source obtains connections. If you do not specify a fully qualified path name, Application Server uses the default location of `WAS_HOME/derby` (or the equivalent default for a UNIX or LINUX environment).
     – Example database path name for Windows: `c:\temp\sampleDB`
     – Example database path name for UNIX or LINUX: `/tmp/sampleDB`

     If no database currently exists for the path name you want to specify, simply append `;create=true` to the path name to create a database dynamically. (For example: `c:\temp\sampleDB;create=true`)

3.

   Derby Network Server using Universal JDBC driver -- Deprecated

This JDBC provider is deprecated in WebSphere Application Server Version 6.1. Therefore it is no longer an available choice among provider types in the administrative console.

This Derby driver takes advantage of the Network Server support that the DB2 universal Type 4 JDBC driver provides. You cannot use any Version 4.0 data sources with Cloudscape v10.x.

Use the following one phase data source for the Derby Network Server using the Universal JDBC driver:

com.ibm.db2.jcc.DB2ConnectionPoolDataSource

Requires JDBC driver files:

- **db2jcc.jar** If you install and run DB2, you must use the **db2jcc.jar** file that comes with DB2. To do that, the classpath in the JDBC template for Derby Network Server is set to be:

  ```
  <classpath>${DB2UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc.jar</classpath>

  <classpath>${CLOUDSCAPE_JDBC_DRIVER_PATH}/otherJars/db2jcc.jar</classpath>

  <classpath>${UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc_license_cu.jar</classpath>
  ```

  which means that the db2jcc.jar from DB2 always takes precedence. Note that this also means that you must set the DB2 environment variable **DB2UNIVERSAL_JDBC_DRIVER_PATH** in WebSphere Application Server when you set up your DB2 data source. This is instead of hard coding the path of the db2jcc.jar for DB2 data sources.
- **db2jcc_license_cu.jar** This file is the DB2 Universal JDBC license file that provides access to the Derby databases using the **Network Server** framework. Use this file to gain access to the database. This file ships with WebSphere and is located in **${UNIVERSAL_JDBC_DRIVER_PATH}**.

  Note: **UNIVERSAL_JDBC_DRIVER_PATH** is a WebSphere environment variable that is already mapped to the location in Websphere Application Server where the license jar file is located, and will only be used if the **DB2UNIVERSAL_JDBC_DRIVER_PATH** is not set. DB2 users should ensure that **DB2UNIVERSAL_JDBC_DRIVER_PATH** is set to avoid loading multiple versions of the db2jcc.jar file.

  Note: **DB2UNIVERSAL_JDBC_DRIVER_PATH** is a WebSphere environment variable that you must set to point to the location of db2jcc.jar file (that comes with DB2). This variable is set only if you create a DB2 provider.

  Note: Derby requires only db2jcc_license_c.jar; however, WebSphere Application Server uses db2jcc_license_cu.jar because this works for both DB2 UDB and Derby.

Requires **DataStoreHelper** class:

`com.ibm.websphere.rsadapter.DerbyNetworkServerDataStoreHelper`

Requires a valid authentication alias.

Requires properties:

- **databaseName** The name of the database from which the data source obtains connections. If you do not specify a fully qualified path name, Application Server uses the default location of `WAS_HOME/derby` (or the equivalent default for a UNIX or LINUX environment).
  - Example database path name for Windows: `c:\temp\sampleDB`
  - Example database path name for UNIX or LINUX: `/tmp/sampleDB`

  If no database currently exists for the path name you want to specify, simply append `;create=true` to the path name to create a database dynamically. (For example: `c:\temp\sampleDB;create=true`)
- **driverType** Only the Type 4 driver is allowed.
- **serverName** The TCP/IP address or the host name for the Distributed Relational Database Architecture (DRDA) server.
- **portNumber** The TCP/IP port number where the DRDA server resides. The default value is port ***1527***.

- **retrieveMessagesfromServerOnGetMessage** This property is required by WebSphere Application Server, not the database. The default value is ***false***. You must set the value of this property to ***true***, to enable text retrieval using the **SQLException.getMessage()** method.

4. **Derby Network Server using Derby Client provider**

   Use this provider to access only Cloudscape 10.1.x databases that run on the same node as WebSphere Application Server.

   Use the following one phase data source for the Derby Network Server using Derby Client provider:

   org.apache.derby.jdbc.ClientConnectionPoolDataSource

   Cloudscape v10.1.x does not support Version 4.0 data sources.

   Requires the following JDBC driver file:
   - **derbyclient.jar**
   -

   Requires **DataStoreHelper** class:

   `com.ibm.websphere.rsadapter.DerbyNetworkServerDataStoreHelper`

   Requires the **databaseName** property: The name of the database from which the data source obtains connections. If you do not specify a fully qualified path name, Application Server uses the default location of `WAS_HOME/derby` (or the equivalent default for a UNIX or LINUX environment).
   - Example database path name for Windows: `c:\`*`temp`*`\`*`sampleDB`*
   - Example database path name for UNIX or LINUX: */`*`tmp`*`/`*`sampleDB`*

   If no database currently exists for the path name you want to specify, append `;create=true` to the path name to create a database dynamically. (For example: `c:\`*`temp`*`\`*`sampleDB`*`;create=true`)

5. **Derby Network Server using Derby Client provider (XA)**

   Use this provider to access only Cloudscape 10.1.x databases that run on the same node as WebSphere Application Server.

   Use the following XA data source for this Derby Network Server using Derby Client provider:

   org.apache.derby.jdbc.XADataSource

   Cloudscape v10.1.x does not support Version 4.0 data sources.

   Requires the following JDBC driver file:
   - **derbyclient.jar**
   -

   Requires **DataStoreHelper** class:

   `com.ibm.websphere.rsadapter.DerbyNetworkServerDataStoreHelper`

   Requires the **databaseName** property: The name of the database from which the data source obtains connections. If you do not specify a fully qualified path name, Application Server uses the default location of `WAS_HOME/derby` (or the equivalent default for a UNIX or LINUX environment).
   - Example database path name for Windows: `c:\`*`temp`*`\`*`sampleDB`*
   - Example database path name for UNIX or LINUX: */`*`tmp`*`/`*`sampleDB`*

   If no database currently exists for the path name you want to specify, append `;create=true` to the path name to create a database dynamically. (For example: `c:\`*`temp`*`\`*`sampleDB`*`;create=true`)

For more information on the new Cloudscape code base, visit the Apache Derby Project Web site at: http://incubator.apache.org/derby/.

Informix

1. **Informix JDBC Driver**

   The Informix JDBC Driver is a Type 4 JDBC driver that provides JDBC access to the Informix database.

   Informix JDBC Driver supports one phase data source:

   com.informix.jdbcx.IfxConnectionPoolDataSource

Requires JDBC driver files:

ifxjdbc.jar
ifxjdbcx.jar

Requires **DataStoreHelper** class:

`com.ibm.websphere.rsadapter.InformixDataStoreHelper`

Requires a valid authentication alias.

Requires properties:

- **serverName** The name of the Informix instance on the server. Example: *ol_myserver*.
- **portNumber** The port on which the instances listen. Example: *1526*.
- **ifxIFXHOST** Either the IP address or the host name of the machine that is running the Informix database to which you want to connect. Example: *myserver.mydomain.com*.

  To support IPv6: On AIX and Solaris, IBM Informix Dynamic Server 10.00 with fix pack 1 supports the IPv6 standard. To enable IPv6 on your WebSphere Application Server connection with one of these Informix releases, input your full IPv6 host name for the ifxIFXHOST property.
- **databaseName** The name of the database from which the data source obtains connections. Example: *Sample*.
- **informixLockModeWait** Although not required, this property enables you to set the number of seconds that Informix software waits for a lock. By default, Informix code throws an exception if it cannot immediately acquire a lock. Example: *2*.

2. **Informix JDBC Driver (XA)**

   The Informix JDBC Driver (XA) is a Type 4 JDBC driver that provides XA-compliant JDBC access to the Informix database.

   Informix JDBC Driver (XA) supports two phase data source:

   com.informix.jdbcx.IfxXADataSource

   Requires JDBC driver files:

   ifxjdbc.jar
   ifxjdbcx.jar

   To use SQLJ: This provider also requires driver file **ifxsqlj.jar** if you plan to use SQLJ for queries.

   Requires **DataStoreHelper** class:

   `com.ibm.websphere.rsadapter.InformixDataStoreHelper`

   Requires a valid authentication alias.

   Requires properties:

   - **serverName** The name of the Informix instance on the server. Example: *ol_myserver*.
   - **portNumber** The port on which the instances listen. Example: *1526*.
   - **ifxIFXHOST** Either the IP address or the host name of the machine that is running the Informix database to which you want to connect. Example: *myserver.mydomain.com*.

     To support IPv6: On AIX and Solaris, IBM Informix Dynamic Server 10.00 with fix pack 1 supports the IPv6 standard. To enable IPv6 on your WebSphere Application Server connection with one of these Informix releases, input your full IPv6 host name for the ifxIFXHOST property.
   - **databaseName** The name of the database from which the data source obtains connections. Example: *Sample*.
   - **ifxIFX_XASPEC** Turn on this property when multiple users access the same database. Activating the property enforces tight coupling of XA transactions within the same global transaction ID, and requires the transactions to share lock space. These parameters help prevent transaction management errors from occurring in cases of multiple client requests. Turn on the `ifxIFX_XASPEC` property by assigning it the value of `Y` or `y`; either character works because the setting is not case-specific. Turn the property off by assigning it the value of `N` or `n`. WebSphere Application Server ignores all other values. Your setting for the property overrides the Informix database system setting.

- **informixLockModeWait** Although not required, this property enables you to set the number of seconds that Informix software waits for a lock. By default, Informix code throws an exception if it cannot immediately acquire a lock. Example: *2*.

For more information on Informix, visit the Informix Web site at: http://www.ibm.com/software/data/informix/

Sybase

1. **Sybase jConnect for JDBC driver**

   The Sybase jConnect JDBC driver is a Type 4 JDBC driver that provides JDBC access to the Sybase database.

   Sybase jConnect JDBC driver supports one phase data source:

   com.sybase.jdbc2.jdbc.SybConnectionPoolDataSource

   Requires JDBC driver files: **jconn2.jar**.

   For IPv6 support: For applications that access Sybase in an IPv6 environment, you must use the Sybase jConnect JDBC driver version 6.0 EBF 12884. This implementation uses different classes and therefore requires a different JAR file and a different implementation class designation. You can define the jConnect JDBC driver v6.0 EBF 12884 in the administrative console by selecting **User-defined** as the database type on the New JDBC provider page. On the JDBC provider general configuration page, replace the default JAR file name with **jconn3.jar**. For the implementation class, input **com.sybase.jdbc3.jdbc.SybConnectionPoolDataSource**. The data store helper class and required properties are the same for all Sybase JDBC drivers.

   Requires **DataStoreHelper** class:

   `com.ibm.websphere.rsadapter.SybaseDataStoreHelper`

   Requires a valid authentication alias.

   Requires properties:
   - **serverName** The name of the database server. Example: *myserver.mydomain.com*.
   - **databaseName** The name of the database from which the data source obtains connections. Example: *Sample*.
   - **portNumber** The TCP/IP port number through which all communications to the server take place. Example: *4100*.
   - **connectionProperties** A custom property required for applications containing EJB 2.0 enterprise beans. Value: `SELECT_OPENS_CURSOR=true`(Type: java.lang.String)

2. **Sybase jConnect for JDBC driver (XA)**

   The Sybase jConnect JDBC driver (XA) is a Type 4 JDBC driver that provides XA-compliant JDBC access to the Sybase database.

   Sybase jConnect JDBC driver (XA) supports two phase data source:

   com.sybase.jdbc2.jdbc.SybXADataSource

   Requires JDBC driver files: **jconn2.jar**.

   For IPv6 support: For applications that access Sybase in an IPv6 environment, you must use the Sybase jConnect JDBC driver version 6.0 EBF 12884. This implementation uses different classes and therefore requires a different JAR file and a different implementation class designation. You can define the jConnect JDBC driver v6.0 EBF 12884 in the administrative console by selecting **User-defined** as the database type on the New JDBC provider page. On the JDBC provider general configuration page, replace the default JAR file name with **jconn3.jar**. For the implementation class, input **com.sybase.jdbc3.jdbc.SybXADataSource**. The data store helper class and required properties are the same for all Sybase JDBC drivers.

   Requires **DataStoreHelper** class:

`com.ibm.websphere.rsadapter.SybaseDataStoreHelper`

Requires a valid authentication alias.

Requires properties:

- **serverName** The name of the database server. Example: *myserver.mydomain.com*
- **databaseName** The name of the database from which the data source obtains connections. Example: *Sample*.
- **portNumber** The TCP/IP port number through which all communications to the server take place. Example: *4100*.
- **connectionProperties** A custom property required for applications containing EJB 2.0 enterprise beans. Value: `SELECT_OPENS_CURSOR=true`(Type: java.lang.String)

For more information on Sybase, visit the Sybase Web site at: http://www.sybase.com/

Oracle

1. **Oracle JDBC Driver**

   The Oracle JDBC Driver provides JDBC access to the Oracle database. This JDBC driver supports both Type 2 JDBC access and Type 4 JDBC access.

   Oracle JDBC Driver supports one phase data source:

   `oracle.jdbc.pool.OracleConnectionPoolDataSource`

   Requires JDBC driver files: **`ojdbc14.jar`**. (Note: If you require Oracle trace, use **`ojdbc14_g.jar`**.)

   Requires **`DataStoreHelper`** class:

   `com.ibm.websphere.rsadapter.OracleDataStoreHelper`

   (Note: If you are running Oracle10g, use `com.ibm.websphere.rsadapter.Oracle10gDataStoreHelper`.)

   Requires a valid authentication alias.

   Requires properties:

   - **URL** The URL that indicates the database from which the data source obtains connections. Example: *jdbc:oracle:thin:@myServer:1521:myDatabase*, where *myServer* is the server name, *1521* is the port it is using for communication, and *myDatabase* is the database name.

2. **Oracle JDBC Driver (XA)**

   The Oracle JDBC Driver (XA) provides XA-compliant JDBC access to the Oracle database. This JDBC driver supports both Type 2 JDBC access and Type 4 JDBC access.

   Oracle JDBC Driver (XA) supports two phase data source:

   `oracle.jdbc.xa.client.OracleXADataSource`

   Requires JDBC driver files: **`ojdbc14.jar`**. (Note: If you require Oracle trace, use **`ojdbc14_g.jar`**.)

   Requires **`DataStoreHelper`** class:

   `com.ibm.websphere.rsadapter.OracleDataStoreHelper`

   (Note: If you are running Oracle10g, use `com.ibm.websphere.rsadapter.Oracle10gDataStoreHelper`.)

   Requires a valid authentication alias.

   Requires properties:

   - **URL** The URL that indicates the database from which the data source obtains connections. Example: *jdbc:oracle:thin:@myServer:1521:myDatabase*, where *myServer* is the server name, *1521* is the port it is using for communication, and *myDatabase* is the database name.

For more information on Oracle, visit the Oracle Web site at: http://www.oracle.com/

MS SQL Server

MS SQL Server 2005: WebSphere Application Server Version 6.10 supports JDBC transactions with the new Microsoft SQL Server 2005, as well as Microsoft SQL Server 2000. For access to either version of the database, use Version 3.5, Service pack 2, of either of the following JDBC providers:

- DataDirect ConnectJDBC Provider, type 4 driver, for MS SQL Server -- including the XA implementation
- IBM WebSphere embedded ConnectJDBC Provider for MS SQL Server -- including the XA implementation

These JDBC drivers provide the same function for MS SQL Server 2005 as MS SQL Server 2000. As long as you use only those new features of MS SQL Server 2005 that have no impact on JDBC transactions, you generally risk no exceptions by upgrading to the new version. WebSphere Application Server does, however, support two new options for setting isolation level in MS SQL Server 2005: SNAPSHOT and READ_COMMITTED_SNAPSHOT. The following chart describes these isolation levels as well as the few requirements and concerns for using MS SQL Server 2005 with Application Server v6.1.

Compatibility or New Usage concern		Resolution or Recommended action
New requirements:	Specifying your database version	You do not have to specify your version of MS SQL Server. Specify the name of your database as usual. WebSphere Application Server detects the version and makes any necessary class attribute adjustments.

	Parenthesis requirement for locking hints	MS SQL Server 2005 requires that you place parentheses around locking hints. For example:
		`select value from t1 (holdlock)`
		`where name = ?`

	New syntax for joining multiple locking hints	• MS SQL Server 2005 requires use of the keyword **with** to join multiple locking hints. For example:
		`select value from t1 with (updlock rowlock)`
		`where name = ?`
		• For MS SQL Server 2000, no keyword is required. For example:
		`select value from t1 (updlock rowlock)`
		`where name = ?`

	Use of the `alternate servers` custom data source property	Verify that all application component clients of the data source issue commands that are valid for both database versions before you configure the `alternate servers` property to include both MS SQL Server 2005 and Server 2000 machines.
	Deprecated data types	Microsoft deprecated three data types for SQL Server 2005, which are shown in the following list along with each replacement data type: • `text`, replaced by `varchar(max)` • `ntext`, replaced by `nvarchar(max)` • `image`, replaced by `varbinary(max)`

Compatibility or New Usage concern		Resolution or Recommended action
New features:	SNAPSHOT isolation level	This new isolation level implements optimistic locking for transactions in which MS SQL Server 2005 serializes the data. You must configure the ALLOW_SNAPSHOT_ISOLATION setting on the database, and then set the isolation level in one of two ways: • By isolation level constant: Invoke the method **setTransactionIsolation** with one of three new attributes: – `conn.setTransactionIsolation (com.ibm.websphere.jdbc.extensions. ExtConstants.TRANSACTION_SNAPSHOT)` – `conn.setTransactionIsolation (com.ddtek.jdbc.extensions. ExtConstants.TRANSACTION_SNAPSHOT)` – `conn.setTransactionIsolation(16)` • By custom data source property: – Set the new data source custom property **snapshotSerializable** to true, ***and*** – Invoke the method **setTransactionIsolation** with the attribute: `conn.setTransactionIsolation (java.sql.Connection.TRANSACTION_SERIALIZABLE)`
	READ_COMMITTED_SNAPSHOT isolation level	This isolation level is a new implementation of Read committed. The policy enforces optimistic locking for read operations with MS SQL Server 2005. You must configure the isolation level on the database. Then invoke the method **setTransactionIsolation** with the attribute: `conn.setTransactionIsolation (java.sql.Connection.TRANSACTION_READ_COMMITTED)`
	Transact-SQL enhancements, including: new functions, additional data types, and the ability to create recursive queries	WebSphere Application Server does not currently support these features; do not use them with WebSphere Application Server Version 6.10.

Consult the Microsoft Web page at http://msdn2.microsoft.com/en-us/library/ms143232(en-US.SQL.90).aspx for a complete list of deprecated items, as well as backward compatibility provisions, for MS SQL Server 2005.

For the MS SQL Server JDBC drivers that support both database versions, perform the same steps and set the same class paths and properties that were previously required for MS SQL Server 2000.

1. **DataDirect ConnectJDBC type 4 driver for MS SQL Server**

   DataDirect ConnectJDBC type 4 driver for MS SQL Server is a Type 4 JDBC driver that provides JDBC access to the MS SQL Server 2005 and MS SQL Server 2000 databases. This provider is for use only with the Connect JDBC driver purchased from DataDirect Technologies.

   This JDBC provider supports this data source:

   com.ddtek.jdbcx.sqlserver.SQLServerDataSource

   Requires JDBC driver files:

   sqlserver.jar,
   base.jar and **util.jar**

(The **spy.jar** file is optional. You need this file to enable spy logging. The **spy.jar** file is not in the same directory as the other three jar files. Instead, it is located in the **../spy/** directory.)

Requires **DataStoreHelper** class:

com.ibm.websphere.rsadapter.ConnectJDBCDataStoreHelper

Requires a valid authentication alias.

Requires properties:

- **serverName** The name of the server in which MS SQL Server resides. Example: *myserver.mydomain.com*
- **portNumber** The TCP/IP port that MS SQL Server uses for communication. Port 1433 is the default.
- **databaseName** The name of the database from which the data source obtains connections. Example: *Sample*.

2. **DataDirect ConnectJDBC type 4 driver for MS SQL Server (XA)**

   DataDirect ConnectJDBC type 4 driver for MS SQL Server (XA) is a Type 4 JDBC driver which provides XA-compliant JDBC access to the MS SQL Server 2005 and MS SQL Server 2000 databases. This provider is for use only with the Connect JDBC driver purchased from DataDirect Technologies.

   This JDBC provider supports this data source:

   com.ddtek.jdbcx.sqlserver.SQLServerDataSource.

   Requires JDBC driver files:

   sqlserver.jar,
   base.jar and **util.jar.**

   (The **spy.jar** file is optional. You need this file to enable spy logging. The **spy.jar** file is not in the same directory as the other three jar files. Instead, it is located in the **../spy/** directory.)

   Requires **DataStoreHelper** class:

   com.ibm.websphere.rsadapter.ConnectJDBCDataStoreHelper

   Requires a valid authentication alias.

   Requires properties:

   - **serverName** The name of the server in which MS SQL Server resides. Example: *myserver.mydomain.com*
   - **portNumber** The TCP/IP port that MS SQL Server uses for communication. Port 1433 is the default.
   - **databaseName** The name of the database from which the data source obtains connections. Example: *Sample*.

   For more information on the DataDirect ConnectJDBC driver, visit the DataDirect Web site at: http://www.datadirect-technologies.com/

3. **IBM WebSphere embedded ConnectJDBC driver for MS SQL Server**

   WebSphere embedded ConnectJDBC driver for MS SQL Server is a Type 4 JDBC driver that provides JDBC access to the MS SQL Server 2005 and MS SQL Server 2000 databases. This JDBC driver ships with WebSphere Application Server. Only use this provider with the Connect JDBC driver embedded in WebSphere; it cannot be used with a Connect JDBC driver purchased separately from DataDirect Technologies.

   This JDBC provider supports this data source:

   com.ibm.websphere.jdbcx.sqlserver.SQLServerDataSource.

   Requires JDBC driver files:

   sqlserver.jar
   base.jar and
   util.jar.

(The **spy.jar** file is optional. You need this file to enable spy logging. The **spy.jar** file for the WebSphere embedded Connect JDBC driver ships with WebSphere Application Server. All the files are located in the `WAS_HOME/lib/` directory.)

Requires **DataStoreHelper** class:

`com.ibm.websphere.rsadapter.WSConnectJDBCDataStoreHelper`

Requires a valid authentication alias.

Requires properties:

- **serverName** The name of the server in which MS SQL Server resides. Example: *myserver.mydomain.com*
- **portNumber** The TCP/IP port that MS SQL Server uses for communication. Port 1433 is the default.
- **databaseName** The name of the database from which the data source obtains connections. Example: *Sample*.

4. **IBM WebSphere embedded ConnectJDBC driver for MS SQL Server (XA)**

   WebSphere embedded ConnectJDBC driver for MS SQL Server (XA) is a Type 4 JDBC driver that supports two-phase commit transactions on connections with the MS SQL Server 2005 and MS SQL Server 2000 databases. This JDBC driver ships with WebSphere Application Server. Use this provider with the IBM WebSphere Connect JDBC driver embedded in WebSphere Application Server. Do not use it with the DataDirect Connect JDBC driver purchased separately from DataDirect Technologies.

   The ConnectJDBC provider supports the following data source:

   com.ibm.websphere.jdbcx.sqlserver.SQLServerDataSource.

   Requires JDBC driver files:

   sqlserver.jar
   base.jar and
   util.jar.

   An additional file, the **spy.jar** file, is optional. You need **spy.jar** for spy logging, which is a form of JDBC driver-level trace.

   All of the JAR files in the previous list are shipped with WebSphere Application Server and are installed automatically with the product. They are also updated automatically when you apply WebSphere Application Server service packs.

   Requires **DataStoreHelper** class:

   `com.ibm.websphere.rsadapter.WSConnectJDBCDataStoreHelper`

   Requires a valid authentication alias.

   Requires properties:

   - **serverName** The name of the server in which MS SQL Server resides. Example: *myserver.mydomain.com*
   - **portNumber** The TCP/IP port that MS SQL Server uses for communication. Port 1433 is the default.
   - **databaseName** The name of the database from which the data source obtains connections. Example: *Sample*.

   Patches to the IBM WebSphere Connect JDBC driver jar files are installed automatically when you apply WebSphere Application Server service packs. However, to update Microsoft SQL Server-side programs for this JDBC driver, you must go to the IBM FTP site for WebSphere Application Server embedded product updates. Use the following URL:

   ftp://ftp.software.ibm.com/software/websphere/info/tools/DataDirect/datadirect.htm

   An important server-side program is Stored Procedures for the Java Transaction API (JTA). Whether you need to run one or two phase transactions with the XA-enabled IBM WebSphere Connect JDBC driver, you must install Stored Procedures for JTA on all machines that run Microsoft SQL. The WebSphere Application Server installation disks contain a base level of Stored Procedures for JTA. Go to the previously listed FTP site for updates to this API.

   Install Stored Procedures for JTA by performing the following steps:

a. Determine whether you are running the 32-bit or 64-bit MS SQL Server and select the appropriate `sqljbc.dll` and `instjdbc.sql` files.
b. Stop your MS SQL Server service.
c. Copy the `sqljdbc.dll` file into your `%SQL_SERVER_INSTALL%\Binn\` directory.
d. Restart the MS SQL Server service.
e. Run the `instjdbc.sql` script. (The script can be run by the MS SQL Server Query Analyzer or the ISQL utility).

`ftp://ftp.software.ibm.com/software/websphere/info/tools/DataDirect/datadirect.htm`

5. **DataDirect SequeLink type 3 JDBC driver for MS SQL Server** -- Deprecated

   This type 3 JDBC driver for MS SQL Server is deprecated in WebSphere Application Server Version 6.0. Therefore it is no longer an available choice among provider types in the administrative console.

   For best results with WebSphere Application Server JDBC access to MS SQL Server, use only JDBC drivers that are *not* marked for deprecation. However, if you must continue using a deprecated driver for JDBC access to MS SQL Server, you can configure it through the WebSphere Application Server administrative console. Be sure to select **User-defined** for the database type. This selection triggers the console to display default class files, data source interfaces, and so on for your user-defined JDBC provider type. Replace those defaults with the following settings that are specific to the DataDirect SequeLink type 3 JDBC driver. For information on this task, read the Configuring a JDBC provider using the administrative console section of the *Administering applications and their environment* PDF book.

   Incompatible with MS SQL Server 2005: Use this JDBC driver for access to MS SQL Server 2000 only.

   DataDirect SequeLink type 3 JDBC driver supports the following data source:

   `com.ddtek.jdbcx.sequelink.SequeLinkDataSource`

   Requires JDBC driver files:

   `sljc.jar` and
   `spy-sl.jar`

   (The JDBC driver shipped with WebSphere Application Server requires the **`sljc.jar`** and the **`spy-sl.jar`** files. The JDBC driver purchased from DataDirect requires the **`sljc.jar`** and the **`spy.jar`** files. The **`spy.jar`** and **`spy-sl.jar`** files are optional. You need these files to enable spy logging.)

   Requires **`DataStoreHelper`** class:

   `com.ibm.websphere.rsadapter.SequeLinkDataStoreHelper`

   Requires a valid authentication alias.

   Requires properties:
   - **serverName** The name of the server in which SequeLink Server resides. Example: *myserver.mydomain.com*
   - **portNumber** The TCP/IP port that SequeLink Server uses for communication. By default, SequeLink Server uses port 19996.
   - **databaseName** The name of the database from which the data source obtains connections. Example: *Sample*.
   - **enable2phase** This property is necessary only if your application must participate in two-phase transactions. By default, Application Server sets a user-defined implementation type to a connection pool data source. The connection pool data source supports only one-phase transactions. Configure two-phase transaction support by setting the *enable2Phase* custom property on each data source that you create with your user-defined JDBC provider. Follow these steps:
     a. Go to the Custom properties administrative console page by clicking **Resources** > **JDBC Providers** > *your_JDBC_provider* > **Data sources** > *your_data_source* > **Custom properties**.
     b. Click **New**.
     c. Input `enable2Phase` as the property name and assign it the value of `true`. For Version 4 data sources, use a property that works in the opposite manner: Input `disable2Phase` and assign it the value of `false`.

The DataDirect SequeLink type 3 JDBC driver requires installation of SequeLink Server on all machines running MS SQL Server. See the readme.html file found in the DataDirect folder on the WebSphere Application Server CD for instructions on how to install SequeLink Server. (Install SequeLink Server from the WebSphere Application Server CD only if you are using the SequeLink JDBC driver embedded in WebSphere. Otherwise, install a copy of SequeLink Server purchased from DataDirect Technologies.)

Patches to the IBM WebSphere SequeLink JDBC driver jar files are installed automatically when applying WebSphere Application Server service packs. If updates are ever needed for the Microsoft SQL Server-side installables (SequeLink server) for the IBM WebSphere SequeLink JDBC driver, they will be made available from the following FTP site:

`ftp://ftp.software.ibm.com/software/websphere/info/tools/DataDirect/datadirect.htm`

For more information on the DataDirect SequeLink type 3 JDBC driver, visit the DataDirect Web site at:

`http://www.datadirect-technologies.com/`

6. **Microsoft JDBC driver for MS SQL Server 2000** -- Deprecated

   This type 4 JDBC driver for MS SQL Server 2000 is deprecated in WebSphere Application Server Version 6.0. Therefore it is no longer an available choice among provider types in the administrative console.

   For best results with WebSphere Application Server JDBC access to MS SQL Server, use only JDBC drivers that are *not* marked for deprecation. However, if you must continue using a deprecated driver for JDBC access to MS SQL Server, you can configure it through the WebSphere Application Server administrative console. Be sure to select **User-defined** for the database type. This selection triggers the console to display default class files, data source interfaces, and so on for your user-defined JDBC provider type. Replace those defaults with the following settings that are specific to the Microsoft JDBC driver for MS SQL Server 2000. Follow the steps listed in Configuring a JDBC provider using the administrative console section of the *Administering applications and their environment* PDF book..

   Microsoft JDBC driver for MS SQL Server 2000 supports the following data source:

   `com.microsoft.jdbcx.sqlserver.SQLServerDataSource`

   Requires JDBC driver files:

   `mssqlserver.jar`,
   `msbase.jar` and **`msutil.jar`**

   (The **`spy.jar`** file is optional. You need it to enable spy logging. However, Microsoft does not ship the **`spy.jar`** file. Contact Microsoft about this issue.)

   Requires **`DataStoreHelper`** class:

   `com.ibm.websphere.rsadapter.ConnectJDBCDataStoreHelper`

   Requires a valid authentication alias.

   Requires properties:
   - **serverName** The name of the server in which MS SQL Server resides. Example: *myserver.mydomain.com*
   - **portNumber** The TCP/IP port that MS SQL Server uses for communication. Port 1433 is the default.
   - **databaseName** The name of the database from which the data source obtains connections. Example: *Sample*.
   - **enable2phase** This property is necessary only if your application must participate in two-phase transactions. By default, Application Server sets a user-defined implementation type to a connection pool data source. The connection pool data source supports only one-phase transactions. Configure two-phase transaction support by setting the *enable2Phase* custom property on each data source that you create with your user-defined JDBC provider. Follow these steps:
     a. Go to the Custom properties administrative console page by clicking **Resources** > **JDBC Providers** > *your_JDBC_provider* > **Data sources** > *your_data_source* > **Custom properties**.
     b. Click **New**.
     c. Input `enable2Phase` as the property name and assign it the value of `true`. For Version 4 data sources, use a property that works in the opposite manner: Input `disable2Phase` and assign it the value of `false`.

For more information on the Microsoft JDBC driver for MS SQL Server 2000, visit the Microsoft Web site at:

`http://www.microsoft.com/sql`

Example: Using the Java Management Extensions API to create a JDBC driver and data source for container-managed persistence

```
//
// "This program may be used, executed, copied, modified and distributed
// without royalty for the purpose of developing, using, marketing, or
// distributing."
//
// Product 5630-A36,  (C) COPYRIGHT International Business Machines
// Corp., 2001, 2002
// All Rights Reserved * Licensed Materials - Property of IBM
//
import java.util.*;
import javax.sql.*;
import javax.transaction.*;
import javax.management.*;

import com.ibm.websphere.management.*;
import com.ibm.websphere.management.configservice.*;
import com.ibm.ws.exception.WsException;


/**
 * Creates a node scoped resource.xml entry for a DB2 XA datasource.
 * The datasource created is for CMP use.
 *
* We need following to run
 * set classpath=%classpath%;D:\WebSphere\AppServer\lib\wsexception.jar;
      D:\WebSphere\AppServer\lib\wasjmx.jar;D:\$WAS_HOME\lib\wasx.jar
  */
public class CreateDataSourceCMP {

   String dsName = "markSection"; // ds display name , also jndi name and
    CF name
   String dbName = "SECTION";     // database name
   String authDataAlias = "db2admin"; // an authentication data alias
   String uid = "db2admin";           // userid
   String pw  = "db2admin";           //  password
   String dbclasspath = "D:/SQLLIB/java/db2java.zip"; // path to the db driver




   /**
    * Main method.
    */
   public static void main(String[] args) {
      CreateDataSourceCMP cds = new CreateDataSourceCMP();

      try {
         cds.run(args);
      } catch (com.ibm.ws.exception.WsException ex) {
         System.out.println("Caught this " + ex   );
         ex.printStackTrace();
         //ex.getCause().printStackTrace();
      } catch (Exception ex) {
         System.out.println("Caught this " + ex );
         ex.printStackTrace();
      }
   }
```

```
   /**
    * This method creates the datasource using JMX.
* The datasource created here is only written into resources.xml.
    * It is not bound into namespace until the server is restarted, or
        an application started
    */
   public void run(String[] args) throws Exception {

      try {
         // Initialize the AdminClient.
         Properties adminProps = new Properties();
         adminProps.setProperty(AdminClient.CONNECTOR_TYPE,
          AdminClient.CONNECTOR_TYPE_SOAP);
         adminProps.setProperty(AdminClient.CONNECTOR_HOST, "localhost");
         adminProps.setProperty(AdminClient.CONNECTOR_PORT, "8880");
         AdminClient adminClient =
              AdminClientFactory.createAdminClient(adminProps);

         // Get the ConfigService implementation.
         com.ibm.websphere.management.configservice.ConfigServiceProxy
           configService =
         new com.ibm.websphere.management.configservice.
                ConfigServiceProxy(adminClient);

         Session session = new Session();

         // Use this group to add to the node scoped resource.xml.
         ObjectName node1 = ConfigServiceHelper.createObjectName(null,
                "Node", null);
         ObjectName[] matches = configService.queryConfigObjects(session,
              null, node1, null);
         node1 = matches[0];     // use the first node found

         // Use this group to add to the server1 scoped resource.xml.
         ObjectName server1 = ConfigServiceHelper.createObjectName(null,
                 "Server", "server1");
         matches = configService.queryConfigObjects(session, null, server1,
                null);
         server1 = matches[0];   // use the first server found

         // Create the JDBCProvider
         String providerName = "DB2 JDBC Provider (XA)";
         System.out.println("Creating JDBCProvider " + providerName );

         // Prepare the attribute list
         AttributeList provAttrs = new AttributeList();
         provAttrs.add(new Attribute("name", providerName));
         provAttrs.add(new Attribute("implementationClassName",
                 "COM.ibm.db2.jdbc.DB2XADataSource"));
         provAttrs.add(new Attribute("description","DB2 JDBC2-compliant
                      XA Driver"));

         //create it
         ObjectName jdbcProv = configService.createConfigData(session,node1,
                 "JDBCProvider", "resources.jdbc:JDBCProvider",provAttrs);
         // now plug in the classpath
configService.addElement(session,jdbcProv,"classpath",dbclasspath,-1);


// Search for RRA so we can link it to the datasource
         ObjectName rra = ConfigServiceHelper.createObjectName(null,
                 "J2CResourceAdapter", null);
         matches = configService.queryConfigObjects(session, node1, rra,
                  null);
         rra = matches[0]; // use the first J2CResourceAdapter segment for
                  builtin_rra
```

```
        // Prepare the attribute list
        AttributeList dsAttrs = new AttributeList();
        dsAttrs.add(new Attribute("name", dsName));
        dsAttrs.add(new Attribute("jndiName", "jdbc/" + dsName));
        dsAttrs.add(new Attribute("datasourceHelperClassname",
             "com.ibm.websphere.rsadapter.DB2DataStoreHelper"));
        dsAttrs.add(new Attribute("statementCacheSize", new Integer(10)));
        dsAttrs.add(new Attribute("relationalResourceAdapter", rra));
        // this is where we make the link to "builtin_rra"
        dsAttrs.add(new Attribute("description", "JDBC Datasource for
            mark section CMP 2.0 test"));
        dsAttrs.add(new Attribute("authDataAlias",authDataAlias));

        // Create the datasource
        System.out.println("  **  Creating datasource");
        ObjectName dataSource =
            configService.createConfigData(session,jdbcProv,"DataSource",
              "resources.jdbc:DataSource",dsAttrs);

        // Add a propertySet.
        AttributeList propSetAttrs = new AttributeList();
        ObjectName resourcePropertySet =
          configService.createConfigData(session,dataSource,"propertySet",
              "",propSetAttrs);

        // Add resourceProperty databaseName
        AttributeList propAttrs1 = new AttributeList();
        propAttrs1.add(new Attribute("name", "databaseName"));
        propAttrs1.add(new Attribute("type", "java.lang.String"));
        propAttrs1.add(new Attribute("value", dbName));

        configService.addElement(session,resourcePropertySet,
             "resourceProperties",propAttrs1,-1);


// Now Create the corresponding J2CResourceAdapter Connection Factory object.
        ObjectName jra = ConfigServiceHelper.createObjectName(null,
            "J2CResourceAdapter",null);

        // Get all the J2CResourceAdapter, and I want to add my datasource
        System.out.println("  **  Get all J2CResourceAdapter's");
        ObjectName[] jras = configService.queryConfigObjects(session, node1,
               jra, null);

        int i=0;

        for (;i< jras.length;i++) {
           System.out.println(ConfigServiceHelper.getConfigDataType(jras[i])+
                   " " + i + " = "
                              + jras[i].getKeyProperty(SystemAttributes.
                                _WEBSPHERE_CONFIG_DATA_DISPLAY_NAME)
                              + "\nFrom scope ="
                              + jras[i].getKeyProperty(SystemAttributes.
                                _WEBSPHERE_CONFIG_DATA_ID));
           // quit on the first builtin_rra
           if (jras[i].getKeyProperty(SystemAttributes.
                               _WEBSPHERE_CONFIG_DATA_DISPLAY_NAME)
               .equals("WebSphere Relational Resource Adapter")) {
              break;
           }
        }


        if (i >= jras.length) {
           System.out.println("Did not find builtin_rra J2CResourceAdapter
               object creating CF anyways" );
```

```
        } else {
           System.out.println("Found builtin_rra J2CResourceAdapter
                  object at index " + i + " creating CF" );
        }

        // Prepare the attribute list
        AttributeList cfAttrs = new AttributeList();
        cfAttrs.add(new Attribute("name", dsName + "_CF"));
        cfAttrs.add(new Attribute("authMechanismPreference","BASIC_PASSWORD"));
        cfAttrs.add(new Attribute("authDataAlias",authDataAlias));
        cfAttrs.add(new Attribute("cmpDatasource", dataSource ));
        // this is where we make the link to DataSource's xmi:id
        ObjectName cf = configService.createConfigData(session,jras[i],
        "CMPConnectorFactory", "resources.jdbc:CMPConnectorFactory",cfAttrs);


        // ===== start Security section
        System.out.println("Creating an authorization data alias " +
           authDataAlias);

// Find the parent security object
        ObjectName security = ConfigServiceHelper.createObjectName(null,
             "Security", null);
        ObjectName[] securityName = configService.queryConfigObjects(session,
               null, security, null);
        security=securityName[0];

        // Prepare the attribute list
        AttributeList authDataAttrs = new AttributeList();
        authDataAttrs.add(new Attribute("alias", authDataAlias));
        authDataAttrs.add(new Attribute("userId", uid));
        authDataAttrs.add(new Attribute("password", pw));
        authDataAttrs.add(new Attribute("description","Auto created alias
            for datasource"));

        //create it
        ObjectName authDataEntry = configService.createConfigData
           (session,security,"authDataEntries", "JAASAuthData",authDataAttrs);
        // ===== end Security section

        // Save the session
        System.out.println("Saving session" );
        configService.save(session, false);

        // reload resources.xml to bind the new datasource into the name space
        reload(adminClient,true);
     } catch (Exception ex) {
        ex.printStackTrace(System.out);
        throw ex;
     }
  }

  /**
   * Get the DataSourceConfigHelperMbean and call reload() on it
   *
   * @param adminClient
   * @param verbose true - print messages to stdout
   */
  public void reload(AdminClient adminClient,boolean verbose) {
     if (verbose) {
        System.out.println("Finding the Mbean to call reload()");
     }

     // First get the  Mbean
     ObjectName handle = null;
     try {
        ObjectName queryName = new ObjectName("WebSphere:type=
```

```
                DataSourceCfgHelper,*");
         Set s = adminClient.queryNames(queryName, null);
         Iterator iter = s.iterator();
         if (iter.hasNext()) handle = (ObjectName)iter.next();
      } catch (MalformedObjectNameException mone) {
         System.out.println("Check the program variable queryName" + mone);
      } catch (com.ibm.websphere.management.exception.ConnectorException ce) {
         System.out.println("Cannot connect to the application server" + ce);
      }

      if (verbose) {
         System.out.println("Calling reload()");
      }
      Object result = null;
      try {
         result = adminClient.invoke(handle, "reload", new Object[] {},
               new String[] {});
      } catch (MBeanException mbe) {
         if (verbose) {
            System.out.println("\tMbean Exception calling reload" + mbe);
         }
      } catch (InstanceNotFoundException infe) {
         System.out.println("Cannot find reload ");
      } catch (Exception ex) {
         System.out.println("Exception occurred calling reload()"  + ex);
      }

      if (result==null && verbose) {
         System.out.println("OK reload()");
      }
   }

}
```

Example: Using the Java Management Extensions API to create a JDBC driver and data source for bean-managed persistence, session beans, or servlets

```
//
// "This program may be used, executed, copied, modified and distributed without royalty for the
// purpose of developing, using, marketing, or distributing."
//
// Product 5630-A36,  (C) COPYRIGHT International Business Machines Corp., 2001, 2002
// All Rights Reserved * Licensed Materials - Property of IBM
//
import java.util.*;
import javax.sql.*;
import javax.transaction.*;
import javax.management.*;

import com.ibm.websphere.management.*;
import com.ibm.websphere.management.configservice.*;
import com.ibm.ws.exception.WsException;


/**
 * Creates a node scoped resource.xml entry for a DB2 XA datasource.
 * The datasource created is for BMP use.
 *
 * We need following to run
 * set classpath=%classpath%;D:\WebSphere\AppServer\lib\wsexception.jar;
 * D:\WebSphere\AppServer\lib\wasjmx.jar;D:\$WAS_HOME\lib\wasx.jar
 */
public class CreateDataSourceBMP {
```

```
String dsName = "markSection"; // ds display name , also jndi name and CF name
String dbName = "SECTION";     // database name
String authDataAlias = "db2admin"; // an authentication data alias
String uid = "db2admin";           // userid
String pw  = "db2admin";           //  password
String dbclasspath = "D:/SQLLIB/java/db2java.zip"; // path to the db driver



/**
 * Main method.
 */
public static void main(String[] args) {
   CreateDataSourceBMP cds = new CreateDataSourceBMP();

   try {
      cds.run(args);
   } catch (com.ibm.ws.exception.WsException ex) {
      System.out.println("Caught this " + ex   );
      ex.printStackTrace();
      //ex.getCause().printStackTrace();
   } catch (Exception ex) {
      System.out.println("Caught this " + ex );
      ex.printStackTrace();
   }
}


/**
 * This method creates the datasource using JMX.
 *
 * The datasource created here is only written into resources.xml.
 * It is not bound into namespace until the server is restarted, or an application started
 */
public void run(String[] args) throws Exception {

   try {
      // Initialize the AdminClient.
      Properties adminProps = new Properties();
      adminProps.setProperty(AdminClient.CONNECTOR_TYPE, AdminClient.CONNECTOR_TYPE_SOAP);
      adminProps.setProperty(AdminClient.CONNECTOR_HOST, "localhost");
      adminProps.setProperty(AdminClient.CONNECTOR_PORT, "8880");
      AdminClient adminClient = AdminClientFactory.createAdminClient(adminProps);

      // Get the ConfigService implementation.
      com.ibm.websphere.management.configservice.ConfigServiceProxy configService =
      new com.ibm.websphere.management.configservice.ConfigServiceProxy(adminClient);

      Session session = new Session();

      // Use this group to add to the node scoped resource.xml.
      ObjectName node1 = ConfigServiceHelper.createObjectName(null, "Node", null);
      ObjectName[] matches = configService.queryConfigObjects(session, null, node1, null);
      node1 = matches[0];     // use the first node found

      // Use this group to add to the server1 scoped resource.xml.
      ObjectName server1 = ConfigServiceHelper.createObjectName(null, "Server", "server1");
      matches = configService.queryConfigObjects(session, null, server1, null);
      server1 = matches[0];   // use the first server found

      // Create the JDBCProvider
      String providerName = "DB2 JDBC Provider (XA)";
      System.out.println("Creating JDBCProvider " + providerName );

      // Prepare the attribute list
      AttributeList provAttrs = new AttributeList();
```

```
     provAttrs.add(new Attribute("name", providerName));
     provAttrs.add(new Attribute("implementationClassName", "COM.ibm.db2.jdbc.DB2XADataSource"));
     provAttrs.add(new Attribute("description","DB2 JDBC2-compliant XA Driver"));

     //create it
     ObjectName jdbcProv = configService.createConfigData(session,node1,"JDBCProvider",
"resources.jdbc:JDBCProvider",provAttrs);
     // now plug in the classpath
     configService.addElement(session,jdbcProv,"classpath",dbclasspath,-1);




     // Search for RRA so we can link it to the datasource
     ObjectName rra = ConfigServiceHelper.createObjectName(null, "J2CResourceAdapter", null);
     matches = configService.queryConfigObjects(session, node1, rra, null);
     rra = matches[0]; // use the first J2CResourceAdapter segment for builtin_rra

     // Prepare the attribute list
     AttributeList dsAttrs = new AttributeList();
     dsAttrs.add(new Attribute("name", dsName));
     dsAttrs.add(new Attribute("jndiName", "jdbc/" + dsName));
     dsAttrs.add(new Attribute("datasourceHelperClassname","com.ibm.websphere.rsadapter.DB2DataStoreHelper"));
     dsAttrs.add(new Attribute("statementCacheSize", new Integer(10)));
     dsAttrs.add(new Attribute("relationalResourceAdapter", rra));
// this is where we make the link to "builtin_rra"
     dsAttrs.add(new Attribute("description", "JDBC Datasource for mark section CMP 2.0 test"));
     dsAttrs.add(new Attribute("authDataAlias",authDataAlias));

     // Create the datasource
     System.out.println("  **  Creating datasource");
     ObjectName dataSource = configService.createConfigData(session,jdbcProv,"DataSource",
"resources.jdbc:DataSource",dsAttrs);

     // Add a propertySet.
     AttributeList propSetAttrs = new AttributeList();
     ObjectName resourcePropertySet =configService.createConfigData(session,dataSource,
"propertySet","",propSetAttrs);

     // Add resourceProperty databaseName
     AttributeList propAttrs1 = new AttributeList();
     propAttrs1.add(new Attribute("name", "databaseName"));
     propAttrs1.add(new Attribute("type", "java.lang.String"));
     propAttrs1.add(new Attribute("value", dbName));

     configService.addElement(session,resourcePropertySet,"resourceProperties",propAttrs1,-1);


     // ===== start Security section
     System.out.println("Creating an authorization data alias " + authDataAlias);

     // Find the parent security object
     ObjectName security = ConfigServiceHelper.createObjectName(null, "Security", null);
     ObjectName[] securityName = configService.queryConfigObjects(session, null, security, null);
     security=securityName[0];

     // Prepare the attribute list
     AttributeList authDataAttrs = new AttributeList();
     authDataAttrs.add(new Attribute("alias", authDataAlias));
     authDataAttrs.add(new Attribute("userId", uid));
     authDataAttrs.add(new Attribute("password", pw));
     authDataAttrs.add(new Attribute("description","Auto created alias for datasource"));

     //create it
     ObjectName authDataEntry = configService.createConfigData(session,security,"authDataEntries",
"JAASAuthData",authDataAttrs);
```

```
            // ===== end Security section

            // Save the session
            System.out.println("Saving session" );
            configService.save(session, false);

            // reload resources.xml
            reload(adminClient,true);

         } catch (Exception ex) {
            ex.printStackTrace(System.out);
            throw ex;
         }
      }

      /**
       * Get the DataSourceConfigHelperMbean and call reload() on it
       *
       * @param adminClient
       * @param verbose true - print messages to stdout
       */
      public void reload(AdminClient adminClient,boolean verbose) {
         if (verbose) {
            System.out.println("Finding the Mbean to call reload()");
         }

         // First get the  Mbean
         ObjectName handle = null;
         try {
            ObjectName queryName = new ObjectName("WebSphere:type=DataSourceCfgHelper,*");
            Set s = adminClient.queryNames(queryName, null);
            Iterator iter = s.iterator();
            if (iter.hasNext()) handle = (ObjectName)iter.next();
         } catch (MalformedObjectNameException mone) {
            System.out.println("Check the program variable queryName" + mone);
         } catch (com.ibm.websphere.management.exception.ConnectorException ce) {
            System.out.println("Cannot connect to the application server" + ce);
         }

         if (verbose) {
            System.out.println("Calling reload()");
         }
         Object result = null;
         try {
            result = adminClient.invoke(handle, "reload", new Object[] {}, new String[] {});
         } catch (MBeanException mbe) {
            if (verbose) {
               System.out.println("\tMbean Exception calling reload" + mbe);
            }
         } catch (InstanceNotFoundException infe) {
            System.out.println("Cannot find reload ");
         } catch (Exception ex) {
            System.out.println("Exception occurred calling reload()"  + ex);
         }

         if (result==null && verbose) {
            System.out.println("OK reload()");
         }
      }
   }
```

Chapter 21. Messaging resources

Important file for message-driven beans

The *server_name*-durableSubscriptions.ser file in the WAS_HOME/temp directory is important for the operation of the WebSphere Application Server messaging service, so should not be deleted.

If you do need to delete the WAS_HOME/temp directory or other files in it, ensure that you preserve the following file:

server_name-**durableSubscriptions.ser**

You should not delete this file, because the messaging service uses it to keep track of durable subscriptions for message-driven beans. If you uninstall an application that contains a message-driven bean, this file is used to unsubscribe the durable subscription.

Troubleshooting WebSphere messaging

Use this overview task to help resolve a problem that you think is related to the WebSphere Messaging.

To identify and resolve problems that you think are related to WebSphere Messaging, you can use the standard WebSphere Application Server troubleshooting facilities. If you encounter a problem that you think might be related to WebSphere Messaging, complete the following stages. Some problems and their troubleshooting are specific to whether you are using the embedded WebSphere Messaging or WebSphere MQ as the JMS provider.

1. Check for error messages about messaging. For example, check for error messages that indicate a problem with JMS resources.

   Check in the application server's SystemOut log at *was_home*\logs*server*\SystemOut.

   The associated message reference information provides an explanation and any user actions to resolve the problem. For details, see Troubleshooter reference: Messages in the information center.
2. Check for more informational and error messages that might provide a clue to a related problem. For example, if you have problems accessing JMS resources, check for more error messages and extra details about any problem associated with the JMS provider or with the service integration technologies that the default messaging provider uses.

   For messages related to the resource adapter (JMS) of the default messaging provider, look for the prefix: CWSJR. For messages related to service integration technologies, see the related reference topics.

   If your message-driven bean uses WebSphere Application Server version 5 JMS resources, look for the prefixes: MSGS and WMSG.
3. If you suspect that problems might be related to application use of message-driven beans, see "Troubleshooting message-driven beans" on page 317.
4. Check the Release Notes for specific problems and workarounds The section *Possible Problems and Suggested Fixes* of the Release Notes, available from the WebSphere Application Server library web site, is updated regularly to contain information about known defects and their workarounds. Check the latest version of the Release Notes for any information about your problem. If the Release Notes do not contain any information about your problem, you can also search the Technotes database on the WebSphere Application Server web site.
5. Check your JMS resource configurations If the messaging services seem to be running properly, check that the JMS resources have been configured correctly. For example, check that the JMS activation specification against which a message-driven bean is deployed has been configured correctly. For more information about configuring JMS resources, refer to the Using asynchronous messaging topic.
6. Get a detailed exception dump for messaging. If the information obtained in the preceding steps is still inconclusive, you can enable the application server debug trace for the ″Messaging″ group to provide a detailed exception dump.

Troubleshooting WebSphere MQ messaging

These hints include a description of the scenario where a problem occurred, the probably reason for the problem and the recommended solution.

What kind of problem are you seeing?
- javax.jms.JMSException: MQJMS2008: failed to open MQ queue in JVM log.
- SVC: jms.BrokerCommandFailedExceptfailed: 3008.

If you do not see a problem that resembles yours, or if the information provided does not solve your problem, see “Messaging component troubleshooting tips” on page 316.

If you see WebSphere MQ error messages or reason codes in WebSphere Application Server messages and logs, refer to the WebSphere MQ Messages document at http://publibfi.boulder.ibm.com/epubs/pdf/amqzao05.pdf.

If you are still unable to resolve the problem, see “Troubleshooting help from IBM” on page 126.
- “An MDB listener fails to start”
- “Problems running JMS applications with security enabled” on page 311
- “Queue manager fails to stop on Redhat Linux” on page 311
- “Application server fails to start in zh_TW.EUC locale on Solaris” on page 311
- “Server memory consumption and java.lang.OutOfMemoryError exception when processing JMS messages” on page 311
- “TopicConnectionFactory attributes clash error when using ″Basic″ WebSphere MQ broker (MA0C SupportPac broker)” on page 312
- “Message WSEC5061E: The SOAP Body is not signed is issued when running a secured Web services application using JMS transport and WebSphere MQ” on page 312
- “Message MQJMS1006: invalid value for tempQPrefix is issued when trying to use a V5.1 client with a V5 Default Messaging queue connection factory on a V6 application server” on page 313
- “When you use WebSphere MQ as an external JMS provider, messages sent within a user-managed transaction arrive before the transaction commits” on page 313

For information about messaging problems that are specific to WebSphere Application Server nodes, see the following links:

Application Servers support Web site

Tips for troubleshooting WebSphere messaging [Version 5]

An MDB listener fails to start

If an MDB listener deployed against a listener port fails to start, you should see the following message:

```
WMSG0019E: Unable to start MDB Listener {0}, JMSDestination {1} : {2}
```

To troubleshoot the cause of an MDB listener not starting, check the following factors:
- Check that the administrative resources have been configured correctly; for example, use the administrative console to check the listener port properties: Destination JNDI name and Connection factory JNDI name. Check that other properties of the listener port, destination, and connection factory are correct.
- Check that the queue exists and has been added to the JMS server.
- Check that the queue manager and JMS server have started.
- Check that the Remote Queue Manager Listener has started.
- If security is enabled, check that a component-managed authentication alias has been specified on the queue connection factory or topic connection factory used by the message-driven bean. This is not required if security is not enabled.
- Check that the user ID used to start the MDB listener is appropriately authorized.

Problems running JMS applications with security enabled

When trying to run a JMS application with security enabled, you can encounter authentication problems indicated by error messages; for example: `WMSG0019E: Unable to start MDB Listener PSSampleMDB, JMSDestination Sample/JMS/listen : javax.jms.JMSSecurityException:` (this example indicates that the security credentials supplied are not valid).

The problem can be removed by doing one of the following:
- If the authentication mechanism is set to `Application`, the application must supply valid credentials.
- If the authentication mechanism is set to `Container`, configure the JMS ConnectionFactory with a container-managed Authentication Alias, and ensure that the associated username and password are valid.

MQJMS2013 invalid security authentication supplied for MQQueueManager: If using WebSphere MQ as a JMS Provider, with JMS connection using bindings transport mode, and the user specified is not the current logged on user for the WebSphere Application Server process, the JMS bindings authentication by WebSphere MQ generates the error `MQJMS2013 invalid security authentication supplied for MQQueueManager`. If you want to use WebSphere MQ as a JMS Provider with JMS connection using bindings transport mode, set the property **Transport type**=BINDINGS on the WebSphere MQ queue connection factory. You must also choose one of the following options:
- To use security credentials, ensure that the user specified is the currently logged on user for the WebSphere Application Server process.
- Do not specify security credentials. On the WebSphere MQ connection factory, ensure that both the **Component-managed Authentication Alias** and the **Container-managed Authentication Alias** properties are not set.

For more information about messaging security, see the Asynchronous messaging - security considerations section of the *Developing and deploying applications* PDF book.

Queue manager fails to stop on Redhat Linux

When trying to stop an application server on Redhat Linux, the queue manager can hang with a Java core dump, and the last message in the SystemOut.log file is `Stopping Queue manager...`.

This is caused by a known RedHat problem (https://bugzilla.linux.ibm.com/show_bug.cgi?id=2336), that was introduced in libstdc++-2.96-116.7.2 and beyond.

The workaround is to go back to the libstdc++-2.96-108.1 level.

Application server fails to start in zh_TW.EUC locale on Solaris

If you have set the locale to zh_TW.EUC on Solaris, and are using WebSphere MQ as a JMS provider, you can encounter problems that stop application servers starting up.

If you intend using WebSphere MQ as a JMS provider on Solaris, do not set the LANG and LC_ALL variables to zh_TW.EUC (Traditional Chinese locale) to avoid problems when starting application servers. Set the LANG and LC_ALL variables to zh_TW instead of zh_TW.EUC.

Server memory consumption and java.lang.OutOfMemoryError exception when processing JMS messages

Intensive processing of JMS messages using the default JMS provider (for example, significant concurrent processing of large messages) can cause a java.lang.OutOfMemoryError exception and cause the application server to terminate.

Processing of JMS messages by the default provider is performed by a messaging engine within the application server process, and therefore consumes memory from the application server's JVM heap. This is in contrast with Version 5 where the support for the embedded JMS provider run in a separate process.

If the amount of memory available to the application server's JVM heap has not been configured large enough to handle the effect of the number of concurrent producers or consumers of messages and the message size, then a java.lang.OutOfMemoryError exception is thrown and the application server terminates.

Solution: When preparing to deploy applications that process JMS messages using the default messaging provider, you should plan for the potential consumption of the application server's memory for message processing. You should take into account the potential number of concurrent processors or consumers of messages and the message size, then set the size of the application server's JVM heap to handle the effect.

For example, when preparing to deploy a message-driven bean that is to be used to process messages concurrently, you should plan for the potential consumption of the application server's memory by concurrent endpoints. Each endpoint that is concurrently processing a message request adds at least two times the message size to the server's JVM heap and can add more, especially if a two-phase transaction is in place.

You can configure the amount of memory available to the application server's JVM heap by setting the Initial Heap Size and Maximum Heap Size properties of the application server. For example, on the WebSphere administrative console panel: **Servers** → **Application servers** → *server_name* → **Java and Process Management** → **Process Definition** → **Java Virtual Machine**.

You can configure the number of concurrent MDB endpoints that can process messages by setting the Maximum concurrent endpoints property of the activation specification used to deploy the message-driven bean. For example, on the WebSphere administrative console panel: **Resources** → **JMS Providers** → **Default messaging** → **JMS activation specification** → *activationspec_name*.

TopicConnectionFactory attributes clash error when using ″Basic″ WebSphere MQ broker (MA0C SupportPac broker)

When creating a JMS topic subscriber using the WebSphere MQ messaging provider, the following error message occurs in the SystemOut.log file:

```
"WSVR0017E: Error encountered binding the J2EE resource, TopicConnectionFactory, as <JNDI_NAME>
   from file:<RESOURCES_FILE> com.ibm.ws.runtime.component.binder.ResourceBindingException: invalid
   configuration passed to resource binding logic. REASON: Failed to create connection factory:
   Error raised constructing AdminObject, error code:  TopicConnectionFactory attributes clash  :
   TopicConnectionFactory attributes clash"
```

This problem is causes by the configuration of the JMS topic connection factory used to create the subscriber, which specifies a broker version of ″Basic″ and a message selection value of ″Broker″. The ″Basic″ WebSphere MQ broker (MA0C SupportPac broker) does not support ″Broker″ message selection.

Solution: Change the JMS topic connection factory to specify a message selection value of ″Client″, which is the only supported value for the WebSphere MQ Basic broker (MA0C SupportPac broker).

Message "WSEC5061E: The SOAP Body is not signed" is issued when running a secured Web services application using JMS transport and WebSphere MQ

When running a secured Web services application using JMS transport using the WebSphere MQ messaging provider, the following error message occurs in the SystemOut.log file:

```
[9/7/04 12:10:02:895 GMT-06:00] 00000039 enterprise    I   TRAS0014I: The following exception was
   logged WebServicesFault
```

```
 faultCode: {http://schemas.xmlsoap.org/ws/2003/06/secext}FailedCheck
 faultString: WSEC5061E: The SOAP Body is not signed.; null
 faultActor: null
 faultDetail:
       stackTrace: com.ibm.wsspi.wssecurity.SoapSecurityException: WSEC5061E: The SOAP Body is not
          signed.; null
...
```

The problem scenario has WebSphere Application Server security enabled, and one Web services application configured with Web services security has attempted and failed to use the JMS transport to send SOAP requests to the target Web service. The JMS resource is configured with WebSphere MQ using a remote WebSphere MQ server. The queue manager exists on this WebSphere MQ server.

The cause of this problem is another identical application is also running from a different application server but using the same queue manager and same queue name in the same WebSphere MQ server. The request sent from the original application has been processed through the same queue, but to the different application server where security might not have been enabled.

Solution: To avoid this problem, complete the following steps:

1. Create a unique queue manager with a unique port in the WebSphere MQ server.
2. Reconfigure the JMS resources to use the new queue manager and port; for example, by using the WebSphere administrative console to change the properties of the WebSphere MQ queue connection factory. For information about how to perform this task, read the section, Configuring a JMS queue connection factory for WebSphere MQ in the *Administering applications and their environment* PDF book.
3. Rerun the application.

Message “MQJMS1006: invalid value for tempQPrefix” is issued when trying to use a V5.1 client with a V5 Default Messaging queue connection factory on a V6 application server

When trying to use a V5.1 application client to connect to a queue connection factory defined as a ″V5 Default Messaging″ resource on a V6 application server. the following message appears:

```
com.ibm.websphere.naming.CannotInstantiateObjectException: Exception occurred while the JNDI
   NamingManager was processing a javax.naming.Reference object.
Root exception is com.ibm.websphere.naming.CannotInstantiateObjectException: Exception occurred
   while the JNDI NamingManager was processing a javax.naming.Reference object.
Root exception is javax.jms.JMSException: MQJMS1006: invalid value for tempQPrefix:
```

Cause: The application client is using WebSphere MQ JMS client CSD 04 JAR files. WebSphere Application Server Version 6 sets tempQprefix to blank, which cannot be handled by the CSD 04 release of the method setTempqPrefix.

Solution If the application client uses WebSphere embedded messaging JAR files, then apply the WebSphere Embedded Messaging interim fixes for WebSphere Application Server V5.1. If the client uses external WebSphere MQ JMS client JAR files, than apply CSD05.

When you use WebSphere MQ as an external JMS provider, messages sent within a user-managed transaction arrive before the transaction commits

When you use WebSphere MQ as an external JMS provider, messages sent within a user-managed transaction can arrive before the transaction commits. This occurs only when you use WebSphere MQ as an external JMS provider, and you send messages to a WebSphere MQ queue within a user-managed transaction. The message arrives on the destination queue before the transaction commits.

The cause of this problem is that the WebSphere MQ resource manager has not been enlisted in the user-managed transaction.

The solution is to use a container-managed transaction.

Messaging engine start error, ″CWSIS1501E: The data source has produced an unexpected exception

When a messaging engine attempts to start, the following error appears in the log:

```
CWSIS0002E: The messaging  engine encountered an exception while starting. Exception:
   com.ibm.ws.sib.msgstore.PersistenceException: CWSIS1501E: The data source has produced an
   unexpected exception: java.sql.SQLException: <dbname> DSRA0010E: SQL  State = 2E000, Error
   Code = -1,001DSR A0010E: SQL State = 2E000, Error Code = -1,001
```

Cause: The database for the messaging engine could not be found.

Solution: Review the database name on the messaging engine for accuracy and check that the database has been created.

javax.jms.JMSException: MQJMS2008: failed to open MQ queue in JVM log

This error can occur when the MQ queue name is not defined in the internal Java Message Service (JMS) server queue names list. This problem can occur if a WebSphere Application Server queue destination is created, without adding the queue name to the internal JMS server queue names list.

To resolve this problem:
1. Open the WebSphere Application Server administrative console.
2. Click **Servers > Manage Application Servers >** *server_name*> **Server Components > JMS Servers**.
3. Add the queue name to the list.
4. Save the changes and restart the server.

javax.jms.JMSException: MQJMS3024: unable to start MDB listener

This error can occur if an uninitialized client ID is used (a ClientID that is not associated with a durable subscription). To resolve this problem, set the ClientID in *one* of the following three ways:

- As a property of the tcf via jmsadmin (for example alter tcf(myTCF) clientid(myID))
- Programmatically using TopicConnection.setClientID()
- Set the Client ID field on the WebSphere MQ topic connection factory resource (for more information, see the related links in this topic).

SVC: jms.BrokerCommandFailedExceptfailed: 3008

One possible cause for this error is that you logged on to a Windows 2000 system as an administrator.

To correct this problem, log out and log in again as a user, rather than an administrator.

For current information available from IBM Support on known problems and their resolution, see the IBM Support page.

IBM Support has documents that can save you time gathering information needed to resolve this problem. Before opening a PMR, see the IBM Support page.

Errors in messaging

These hints include a description of the scenario where a problem occurred, the probably reason for the problem and the recommended solution.

What kind of problem are you seeing?
- javax.jms.JMSException: MQJMS2008: failed to open MQ queue in JVM log.
- SVC: jms.BrokerCommandFailedExceptfailed: 3008.

If you do not see a problem that resembles yours, or if the information provided does not solve your problem, see “Messaging component troubleshooting tips” on page 316.

If you see WebSphere MQ error messages or reason codes in WebSphere Application Server messages and logs, refer to the WebSphere MQ Messages document at http://publibfi.boulder.ibm.com/epubs/pdf/amqzao05.pdf.

If you are still unable to resolve the problem, see “Troubleshooting help from IBM” on page 126.

javax.jms.JMSException: MQJMS2008: failed to open MQ queue in JVM log

This error can occur when the MQ queue name is not defined in the internal Java Message Service (JMS) server queue names list. This problem can occur if a WebSphere Application Server queue destination is created, without adding the queue name to the internal JMS server queue names list.

To resolve this problem:
1. Open the WebSphere Application Server administrative console.
2. Click **Servers > Manage Application Servers >** *server_name*> **Server Components > JMS Servers**.
3. Add the queue name to the list.
4. Save the changes and restart the server.

javax.jms.JMSException: MQJMS3024: unable to start MDB listener

This error can occur if an uninitialized client ID is used (a ClientID that is not associated with a durable subscription). To resolve this problem, set the ClientID in *one* of the following three ways:

- As a property of the tcf via jmsadmin (for example alter tcf(myTCF) clientid(myID))
- Programmatically using TopicConnection.setClientID()
- Set the Client ID field on the WebSphere MQ topic connection factory resource (for more information, see the related links in this topic).

SVC: jms.BrokerCommandFailedExceptfailed: 3008

One possible cause for this error is that you logged on to a Windows 2000 system as an administrator.

To correct this problem, log out and log in again as a user, rather than an administrator.

For current information available from IBM Support on known problems and their resolution, see the IBM Support page.

IBM Support has documents that can save you time gathering information needed to resolve this problem. Before opening a PMR, see the IBM Support page.

Errors connecting to WebSphere MQ and creating WebSphere MQ queue connection factory

You can receive exception errors when trying to create a MDBListener instance, because the MQ manager userid does not have write access to the `/tmp` directory.

If this problem does not resembles yours, or if the information provided does not solve your problem, see Troubleshooting WebSphere Messaging. If you are still unable to resolve the problem, contact IBM support for further assistance.

The following exception may occur when trying to create the MDBListener instance:

```
6/23/03 22:45:58:232 CDT] 673106a8 MsgListenerPo W WMSG0049E:
Failed to start MDB PSSampleMDB against listener port SamplePubSubListenerPort
[6/23/03 22:47:58:289 CDT] 673106a8 FreePool E J2CA0046E:
Method createManagedConnctionWithMCWrapper caught an exception
```

```
during creation of the ManagedConnection for resource
JMS$SampleJMSQueueConnectionFactory, throwing ResourceAllocationException.
Original exception:  javax.resource.spi.ResourceAdapterInternalException:
 createQueueConnection failed
com.ibm.mq.MQException: MQJE001: An MQException occurred:
Completion Code 2, Reason 2009
MQJE003: IO error transmitting message buffer at
com.ibm.mq.MQManagedConnectionJ11.(MQManagedConnectionJ11.java:239)
```

This problem occurs because the MQ manager userid does not have write access to the /tmp directory. To correct this problem, before you use a Jacl procedure to configure WebSphere Application Server resources and install an application:

1. Ensure that all applications have write access to /tmp directory. Use the chmod 1777 command on the directory if necessary.
2. Create another subdirectory under /tmp (for example, /tmp/mydir). Use this directory as a ″working directory″ for the Jacl.
3. Restart the server.

Applications that use messaging on startup should start successfully.

Messaging component troubleshooting tips

Use these tips to troubleshoot problems with messaging components.

Problems deploying or running applications which use the WebSphere Application Server messaging capabilities

If you are having problems deploying or running applications which use the WebSphere Application Server messaging capabilities, review these sections:

- “Troubleshooting WebSphere messaging” on page 309
- “Troubleshooting message-driven beans” on page 317
- Consult the Troubleshooting service integration technologies page in the information center

On a Linux platform, embedded messaging does not get removed after uninstalling the product

On a Linux platform, you may find that embedded messaging is not removed when you uninstall the product silently using the **SetRetainMQToTrue.active**=false option.

To prevent this problem, either issue the uninstall command from a local machine, or use ssh instead of telnet to logon to the machine where you issue the uninstall command. For details about using the uninstall command to uninstall your product in a distributed or iSeries environment, see the Uninstall command section in the *Installing your application serving environment* PDF book.

To correct this problem after logging in, issue the following command for a real root login: su -

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

IBM Support has documents and tools that can save you time gathering information needed to resolve problems as described in “Troubleshooting help from IBM” on page 126. Before opening a problem report, see the Support page:

- http://www.ibm.com/servers/eserver/support/iseries/software/v5r3/index.html

Troubleshooting message-driven beans

Use this overview task to help resolve a problem that you think is related to message-driven beans.

Message-driven beans support uses the standard WebSphere Application Server troubleshooting facilities. If you encounter a problem that you think might be related to the message-driven beans, complete the following steps.

1. Check for error messages about message-driven beans. For example, check for error messages that indicate a problem with JMS resources, such as activation specifications or listener ports, that are used by message-driven beans.

   Check in the application server's SystemOut log at *was_home*\logs*server*\SystemOut.

   The associated message reference information provides an explanation and any user actions to resolve the problem. For details, see Troubleshooter reference: Messages in the information center.
2. Check for more informational and error messages that might provide a clue to a related problem. For example, if you have problems accessing JMS resources, check for more error messages and extra details about any problem associated with the JMS provider or with the service integration technologies that the default messaging provider uses.

   For messages related to the resource adapter (JMS) of the default messaging provider, look for the prefix: CWSJR. For messages related to service integration technologies, see the related reference topics.

   If your message-driven bean uses WebSphere Application Server version 5 JMS resources, look for the prefixes: MSGS and WMSG.
3. Check the Release Notes for specific problems and workarounds The section *Possible Problems and Suggested Fixes* of the Release Notes, available from the WebSphere Application Server library Web site, is updated regularly to contain information about known defects and their workarounds. Check the latest version of the Release Notes for any information about your problem. If the Release Notes does not contain any information about your problem, you can also search the Technotes database on the WebSphere Application Server Web site.
4. If your message-driven bean is deployed against a lister port, check that message listener service has started. The message listener service is an extension to the JMS functions of the JMS provider. It provides a listener manager that controls and monitors one or more JMS listeners, which each monitor a JMS destination on behalf of a deployed message-driven bean.
5. Check your JMS resource configurations If the messaging services seem to be running properly, check that the JMS resources have been configured correctly. For example, check that the JMS activation specification against which the message-driven bean is deployed has been configured correctly. For more information about configuring JMS resources for message-driven beans, see Administering support for message-driven beans in the *Administering applications and their environment* PDF book.
6. Get a detailed exception dump for messaging. If the information obtained in the preceding steps is still inconclusive, you can enable the application server debug trace for the ″Messaging″ group to provide a detailed exception dump.

Chapter 22. Mail, URLs, and other J2EE resources

Enabling debugger for a mail session

When you need to debug a JavaMail application, you can use the JavaMail debugging feature. Enabling the debugger triggers the JavaMail component of WebSphere Application Server to print the following data to the `stdout` output stream:

- interactions with the mail servers
- properties of the mail session

This output stream is redirected to the `SystemOut.log` file for the specific application server.

The mail debugger functions on a per session basis. To enable the JavaMail debugging feature:

1. Open the administrative console.
2. Click **Resources**>**Mail Providers**>*mail_session*>**Mail Session**>*mail session*.
3. Click **Debug**. Debug is enabled for just that session.
4. Click **Apply** or **OK**.

The following example shows sample JavaMail debugging output:

```
DEBUG: not loading system providers in <java.home>/lib
DEBUG  not loading optional custom providers file: /META-INF/javamail.providers
DEBUG: successfully loaded default providers

DEBUG  Tables of loaded providers
DEBUG: Providers listed by Class Name:
{com.sun.mail.smtp.SMTPTransport=javax.mail.Provider[TRANSPORT,smtp,com.sun.mail.smtp.SMTPTransport,Sun
Microsystems, Inc], com.sun.mail.imap.IMAPStore=javax.mail.Provider[STORE,imap,com.sun.mail.imap.IMAPStore,Sun
Microsystems, Inc], com.sun.mail.pop3.POP3Store=javax.mail.Provider[STORE,pop3,com.sun.mail.pop3.POP3Store,Sun
Microsystems, Inc]}
DEBUG:  Providers Listed By Protocol:
{imap=javax.mail.Provider[STORE,imap,com.sun.mail.imap.IMAPStore,Sun Microsystems,
Inc], pop3=javax.mail.Provider[STORE,pop3,com.sun.mail.pop3.POP3Store,Sun
Microsystems, Inc], smtp=javax.mail.Provider[TRANSPORT,smtp,com.sun.mail.smtp.SMTPTransport,Sun
Microsystems, Inc]}
DEBUG: not loading optional address map file: /META-INF/javamail.address.map
*** In SessionFactory.getObjectInstance,
  The default SessionAuthenticator is based on:
   store_user = john_smith
   store_pw = abcdef
*** In SessionFactory.getObjectInstance, parameters in the new session:
    mail.store.protocol="imap"
    mail.transport.protocol="smtp"
    mail.imap.user="john_smith"
    mail.smtp.host="smtp.coldmail.com"
    mail.debug="true"
    ws.store.password="abcdef"
    mail.from="john_smith@coldmail.com"
    mail.smtp.class="com.sun.mail.smtp.SMTPTransport"
    mail.imap.class="com.sun.mail.imap.IMAPStore"
    mail.imap.host="coldmail.com"
DEBUG: mail.smtp.class property exists and points to com.sun.mail.smtp.SMTPTransport
DEBUG SMTP: useEhlo true, useAuth false
DEBUG: SMTPTransport trying to connect to host "smtp.coldmail.com", port 25

javax.mail.SendFailedException: Sending failed;
      nested exception is:
      javax.mail.MessagingException: Unknown SMTP host: smtp.coldmail.com;
       nested exception is
       java.net.UnknownHostException: smtp.coldmail.com
```

```
at javax.mail.Transport.send0(Transport.java:219)
at javax.mail.Transport.send(Transport.java:81)
at ws.mailfvt.SendSaveTestCore.runAll(SendSaveTestCore.java:48)
at testers.AnyTester.main(AnyTester.java:130)
```

This output illustrates a connection failure to a Simple Mail Transfer Protocol (SMTP) server because a fictitious name, smtp.coldmail.com, is specified as the server name.

The following list provides tips on reading the previous sample of debugger output:

- The lines headed by *DEBUG* are printed by the JavaMail run-time, while the two lines headed by ***** are printed by the WebSphere environment run-time.
- The first two lines say that some configuration files are skipped. At run-time the JavaMail component attempts to load a number of configuration files from different locations. All those files are not required. If a required file cannot be accessed, however, the JavaMail component creates an exception. In this sample, there is no exception and the third line announces that default providers are loaded.
- The next few lines, headed by either *Providers listed by Class Name* or *Providers Listed by Protocols*, show the protocol providers that are loaded. The three providers that are listed are the default protocol providers that come under the WebSphere built-in mail provider. They are the protocols SMTP, IMAP, and POP3, respectively. If you install special protocol providers (or, in JavaMail terminology, service providers) and these providers are used in the current mail session, you see them listed here with the default providers.
- The two lines headed by ***** and the few lines below them are printed by WebSphere Application Server to show the configuration properties of the current mail session. Although these properties are listed by their internal name rather than the name you establish in the administrative console, you can easily recognize the relationships between them. For example, the property *mail.store.protocol* corresponds to the Protocol Name property in the Store Access section of the mail session configuration page.

  Note: Review the listed properties and values to verify that they correspond.
- The few lines above the exception stack show the JavaMail activities when sending a message. First, the JavaMail API recognizes that the transport protocol is set to SMTP and that the provider `com.sun.mail.smtp.SMTPTransport` exists. Next, the parameters used by SMTP, `useEhlo` and `useAuth`, are shown. Finally, the log shows the SMTP provider trying to connect to the mail server `smtp.coldmail.com`.
- Next is the exception stack. This data indicates that the specified mail server either does not exist or is not functioning.

Chapter 23. Security

Object and file security

This topic discusses the various objects and files that contain sensitive information and need to be protected.

Secure integrated file system files

In addition to enterprise beans and servlets, WebSphere Application Server accesses integrated file system stream files. The following files might contain sensitive information. It is recommended that you give these files close consideration to ensure that unauthorized access is not granted.

- In the /properties subdirectory of your profile, the following files can contain user IDs and passwords:
  - sas.client.props
  - soap.client.props
  - sas.stdclient.properties
  - sas.tools.properties
  - wsserver.key

  By default, the /properties subdirectory is located in the *profile_root* directory. Each of the previous files is shipped with *PUBLIC authority set to *EXCLUDE. The QEJBSVR user profile is granted *RW authority to these files. Additional protection is available through password encoding. For more information, see Password encoding and encryption.
- In the /etc subdirectory if your profile, protect all of the key (KDB) files and trust (JKS) files that you create for your WebSphere Application Server profile.

  For the JKS files, the QEJBSVR user profiles should have *R authority and *PUBLIC should have *EXCLUDE authority.

  For the KDB files, the user profile that the Web server is running under should have *RX authority and *PUBLIC should have *EXCLUDE authority.

Secure database resources for WebSphere Application Server

WebSphere Application Server uses tables to persist data for user applications such as enterprise beans persistence and servlet session data. You have several options for controlling which user profiles are allowed access to this user data. For more information, see Database access security.

Secure WebSphere Application Server files

When you enable WebSphere Application Server security, the server user profile and password are placed into server configuration files, which should be maintained in a secure way using operating system security. Additionally, you can password protect some WebSphere Application Server resources. These passwords are also placed in server configuration files. The server automatically encodes passwords to deter casual observation, but password encoding alone is not sufficient protection.

The following files are located in the /config subdirectory of your profile and they can contain user identifiers and passwords:

- cells/*cell_name*/security.xml
- cells/*cell_name*/nodes/*node_name*/resources.xml
- cells/*cell_name*/nodes/*node_name*/servers/*server_name*/server.xml

For example, for the default profile, the *server_name* is server1.

The server user profile and password are used for authenticating the server when it initializes. This authentication is required for the following reasons:

- The user ID and password are used as the system identity for the server when an enterprise bean security is deployed to use SYSTEM_IDENTITY for method delegation. In this case, the user ID and password are used when method calls are made from one enterprise bean to another.
- The user ID and password are used to authenticate servers for inter-server communication. Because security for these files can be compromised, use a non-default user profile for the server identity and password. The default user profile is QEJBSVR. If you use the local OS user registry, you might choose to create and use a user profile that has no special authorities. For more information, see Running application servers under specific user profiles.

Secure user profiles for WebSphere Application Server

When WebSphere Application Server is first installed, by default, it uses the following user profiles:

QEJB This profile provides access to some administrative data, including passwords.

QEJBSVR
This profile provides the context in which your WebSphere Application Server runs. For security or administrative purposes, you might want to create other user profiles under which to run various parts of WebSphere Application Server. For more information, see Running application servers under specific user profiles.

Troubleshooting security configurations

The following topics help to troubleshoot specific problems that are related to configuring and enabling security configurations.

Refer to Security components troubleshooting tips for instructions on how to troubleshoot errors that are related to security.

Refer to SPNEGO TAI troubleshooting tips for instructions on how to troubleshoot errors that are related to diagnosing Simple and Protected GSS-API Negotiation (SPNEGO) trust association interceptor (TAI) problems and exceptions.

- Errors when configuring or enabling security
- Errors after enabling security
- Access problems after enabling security
- Errors after configuring or enabling Secure Sockets Layer
- Single sign-on configuration troubleshooting tips
- Enterprise Identity Mapping troubleshooting tips
- Authorization provider troubleshooting tips
- Password decoding troubleshooting tips

Security components troubleshooting tips

This document explains basic resources and steps for diagnosing security-related issues in WebSphere Application Server.

Basic resources and steps for diagnosing security-related issues in WebSphere Application Server include:

- What “Log files” on page 323 to look at and what to look for in them.
- What to look at and what to look for “Using SDSF” on page 324.
- “General approach for troubleshooting security-related issues” on page 325 to isolating and resolving security problems.
- When and how to “Trace security” on page 329.
- An overview and table of “CSIv2 CORBA minor codes” on page 330.

The following security-related problems are addressed elsewhere in the information center:

- Errors and access problems after enabling security

  After enabling security, a degradation in performance is realized. For more information about using unrestricted policy files, see the Enabling security for the realm section of the *Securing applications and their environment* PDF book.
- Errors after enabling SSL, or SSL-related error messages
- Errors trying to configure and enable security

If none of these steps solves the problem, check to see if the problem is identified and documented using the links in Diagnosing and fixing problems: Resources for learning.

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

V6.0.x For an overview of WebSphere Application Server security components such as Secure Authentication Services (SAS) and how they work in a distributed or an iSeries environment, refer to the *Securing applications and their environment* PDF book.

Important: SAS is supported only between Version 6.0.x and previous version servers that have been federated in a Version 6.1 cell.

Log files

When troubleshooting the security component, browse the Java Virtual Machine (JVM) logs for the server that hosts the resource you are trying to access. The following is a sample of messages you would expect to see from a server in which the security service has started successfully:

```
SASRas        A CWWSA0001I: Security configuration initialized.
SASRas        A CWWSA0002I: Authentication protocol: CSIV2/IBM
SASRas        A CWWSA0003I: Authentication mechanism: SWAM
SASRas        A CWWSA0004I: Principal name: MYHOSTNAME/aServerID
SASRas        A CWWSA0005I: SecurityCurrent registered.
SASRas        A CWWSA0006I: Security connection interceptor initialized.
SASRas        A CWWSA0007I: Client request interceptor registered.
SASRas        A CWWSA0008I: Server request interceptor registered.
SASRas        A CWWSA0009I: IOR interceptor registered.
NameServerImp I CWNMS0720I: Do Security service listener registration.
SecurityCompo A CWSCJ0242A: Security service is starting
UserRegistryI A CWSCJ0136I: Custom Registry:com.ibm.ws.security.registry.nt.
NTLocalDomainRegistryImpl has been initialized
SecurityCompo A CWSCJ0202A: Admin application initialized successfully
SecurityCompo A CWSCJ0203A: Naming application initialized successfully
SecurityCompo A CWSCJ0204A: Rolebased authorizer initialized successfully
SecurityCompo A CWSCJ0205A: Security Admin mBean registered successfully
SecurityCompo A CWSCJ0243A: Security service started successfully
SecurityCompo A CWSCJ0210A: Security enabled true
```

The following is an example of messages from a server which cannot start the security service, in this case because the administrative user ID and password given to communicate with the user registry is wrong, or the user registry itself is down or misconfigured:

```
SASRas        A CWWSA0001I: Security configuration initialized.
SASRas        A CWWSA0002I: Authentication protocol: CSIV2/IBM
SASRas        A CWWSA0003I: Authentication mechanism: SWAM
SASRas        A CWWSA0004I: Principal name: MYHOSTNAME/aServerID
SASRas        A CWWSA0005I: SecurityCurrent registered.
SASRas        A CWWSA0006I: Security connection interceptor initialized.
SASRas        A CWWSA0007I: Client request interceptor registered.
SASRas        A CWWSA0008I: Server request interceptor registered.
```

```
SASRas        A CWWSA0009I: IOR interceptor registered.
NameServerImp I CWNMS0720I: Do Security service listener registration.

SecurityCompo A CWSCJ0242A: Security service is starting
UserRegistryI A CWSCJ0136I: Custom Registry:com.ibm.ws.security.
registry.nt.NTLocalDomainRegistryImpl has been initialized
Authenticatio E CWSCJ4001E: Login failed for badID/<null>
javax.security.auth.login.LoginException: authentication failed: bad user/password
```

The following is an example of messages from a server for which Lightweight Directory Access Protocol (LDAP) has been specified as the security mechanism, but the LDAP keys have not been properly configured:

```
SASRas        A CWWSA0001I: Security configuration initialized.
SASRas        A CWWSA0002I: Authentication protocol: CSIV2/IBM
SASRas        A CWWSA0003I: Authentication mechanism: LTPA
SASRas        A CWWSA0004I: Principal name: MYHOSTNAME/anID
SASRas        A CWWSA0005I: SecurityCurrent registered.
SASRas        A CWWSA0006I: Security connection interceptor initialized.
SASRas        A CWWSA0007I: Client request interceptor registered.
SASRas        A CWWSA0008I: Server request interceptor registered.
SASRas        A CWWSA0009I: IOR interceptor registered.
NameServerImp I CWNMS0720I: Do Security service listener registration.
SecurityCompo A CWSCJ0242A: Security service is starting
UserRegistryI A CWSCJ0136I: Custom Registry:com.ibm.ws.security.registry.nt.
NTLocalDomainRegistryImpl has been initialized
SecurityServe E CWSCJ0237E: One or more vital LTPAServerObject configuration
attributes are null or not available. The attributes and values are password :
LTPA password does exist, expiration time 30, private key <null>, public key <null>,
and shared key <null>.
```

A problem with the Secure Sockets Layer (SSL) configuration might lead to the following message. Ensure that the keystore location and keystore passwords are valid. Also, ensure the keystore has a valid personal certificate and that the personal certificate public key or certificate authority (CA) root has been extracted on put into the truststore.

```
SASRas        A CWWSA0001I: Security configuration initialized.
SASRas        A CWWSA0002I: Authentication protocol: CSIV2/IBM
SASRas        A CWWSA0003I: Authentication mechanism: SWAM
SASRas        A CWWSA0004I: Principal name: MYHOSTNAME/aServerId
SASRas        A CWWSA0005I: SecurityCurrent registered.
SASRas        A CWWSA0006I: Security connection interceptor initialized.
SASRas        A CWWSA0007I: Client request interceptor registered.
SASRas        A CWWSA0008I: Server request interceptor registered.
SASRas        A CWWSA0009I: IOR interceptor registered.
SASRas        E CWWSA0026E: [SecurityTaggedComponentAssistorImpl.register]
Exception connecting object to the ORB.  Check the SSL configuration to ensure
 that the SSL keyStore and trustStore properties are set properly.  If the problem
persists, contact support for assistance. org.omg.CORBA.OBJ_ADAPTER:
ORB_CONNECT_ERROR (5) - couldn't get Server Subcontract  minor code:
4942FB8F  completed: No
```

Using SDSF

When troubleshooting the security component, use System Display and Search Facility (SDSF) to browse logs for the server that hosts the resource you are trying to access. The following sample of messages helps you see from a server in which the security service has started successfully:

```
 +BBOM0001I com_ibm_authMechanisms_type_OID: No OID for this mechanism.
 +BBOM0001I com_ibm_security_SAF_unauthenticated: WSGUEST.
 +BBOM0001I com_ibm_security_SAF_EJBROLE_Audit_Messages_Suppress: 0.
 +BBOM0001I com_ibm_ws_logging_zos_errorlog_format_cbe: NOT SET, 280
 DEFAULT=0.
```

```
+BBOM0001I com_ibm_CSI_performClientAuthenticationRequired: 0.
+BBOM0001I com_ibm_CSI_performClientAuthenticationSupported: 1.
+BBOM0001I com_ibm_CSI_performTransportAssocSSLTLSRequired: 0.
+BBOM0001I com_ibm_CSI_performTransportAssocSSLTLSSupported: 1.
+BBOM0001I com_ibm_CSI_rmiInboundPropagationEnabled: 1.
+BBOM0001I com_ibm_CSI_rmiOutboundLoginEnabled: 0.
+BBOM0001I com_ibm_CSI_rmiOutboundPropagationEnabled: 1.
+BBOM0001I security_assertedID_IBM_accepted: 0.
+BBOM0001I security_assertedID_IBM_sent: 0.
+BBOM0001I security_disable_daemon_ssl: NOT SET, DEFAULT=0.
+BBOM0001I security_sslClientCerts_allowed: 0.
+BBOM0001I security_sslKeyring: NOT SET.
+BBOM0001I security_zOS_domainName: NOT SET.
+BBOM0001I security_zOS_domainType: 0.
+BBOM0001I security_zSAS_ssl_repertoire: SY1/DefaultIIOPSSL.
+BBOM0001I security_EnableRunAsIdentity: 0.
+BBOM0001I security_EnableSyncToOSThread: 0.
+BBOM0001I server_configured_system_name: SY1.
+BBOM0001I server_generic_short_name: BBOC001.
+BBOM0001I server_generic_uuid:  457
*** Message beginning with BBOO0222I apply to Java within ***
*** WebSphere Application Server Security ***
+BBOO0222I: SECJ6004I: Security Auditing is disabled.
+BBOO0222I: SECJ0215I: Successfully set JAAS login provider 631
configuration class to com.ibm.ws.security.auth.login.Configuration.
+BBOO0222I: SECJ0136I: Custom 632
Registry:com.ibm.ws.security.registry.zOS.SAFRegistryImpl has been initialized
+BBOO0222I: SECJ0157I: Loaded Vendor AuthorizationTable: 633
com.ibm.ws.security.core.SAFAuthorizationTableImpl
```

General approach for troubleshooting security-related issues

When troubleshooting security-related problems, the following questions are very helpful:

Does the problem occur when security is disabled?

This question is a good litmus test to determine that a problem is security related. However, just because a problem only occurs when security is enabled does not always make it a security problem. More troubleshooting is necessary to ensure the problem is really security-related.

Did security seem to initialize properly?

A lot of security code is visited during initialization. So you can see problems there first if the problem is configuration related.

The following sequence of messages that are generated in the `SystemOut.log` indicate normal code initialization of an application server. This sequence varies based on the configuration, but the messages are similar:

```
SASRas        A CWWSA0001I: Security configuration initialized.
SASRas        A CWWSA0002I: Authentication protocol: CSIV2/IBM
SASRas        A CWWSA0003I: Authentication mechanism: SWAM
SASRas        A CWWSA0004I: Principal name: BIRKT20/pbirk
SASRas        A CWWSA0005I: SecurityCurrent registered.
SASRas        A CWWSA0006I: Security connection interceptor initialized.
SASRas        A CWWSA0007I: Client request interceptor registered.
SASRas        A CWWSA0008I: Server request interceptor registered.
SASRas        A CWWSA0009I: IOR interceptor registered.
NameServerImp I CWNMS0720I: Do Security service listener registration.
SecurityCompo A CWSCJ0242A: Security service is starting
UserRegistryI A CWSCJ0136I: Custom Registry:com.ibm.ws.security.registry.nt.
NTLocalDomainRegistryImpl has been initialized
SecurityCompo A CWSCJ0202A: Admin application initialized successfully
SecurityCompo A CWSCJ0203A: Naming application initialized successfully
SecurityCompo A CWSCJ0204A: Rolebased authorizer initialized successfully
```

```
SecurityCompo A CWSCJ0205A: Security Admin mBean registered successfully
SecurityCompo A CWSCJ0243A: Security service started successfully

SecurityCompo A CWSCJ0210A: Security enabled true
```

The following sequence of messages generated in the SDSF active log indicate normal code initialization of an application server. Non-security messages have been removed from the sequence that follows. This sequence will vary based on the configuration, but the messages are similar:

```
Trace: 2005/05/06 17:27:31.539 01 t=8E96E0 c=UNK key=P8 (13007002)
  ThreadId: 0000000a
  FunctionName: printProperties
  SourceId: com.ibm.ws390.orb.CommonBridge
  Category: AUDIT
  ExtendedMessage: BBOJ0077I java.security.policy =
         /WebSphere/V6R1M0/AppServer/profiles/default/pr
Trace: 2005/05/06 17:27:31.779 01 t=8E96E0 c=UNK key=P8 (13007002)
  ThreadId: 0000000a
  FunctionName: printProperties
  SourceId: com.ibm.ws390.orb.CommonBridge
  Category: AUDIT
  ExtendedMessage: BBOJ0077I java.security.auth.login.config =
         /WebSphere/V6R1M0/AppServer/profiles/default/pr
Trace: 2005/05/06 17:27:40.892 01 t=8E96E0 c=UNK key=P8 (13007002)
  ThreadId: 0000000a
  FunctionName: com.ibm.ws.security.core.SecurityDM
  SourceId: com.ibm.ws.security.core.SecurityDM
  Category: INFO
  ExtendedMessage: BBOO0222I: SECJ0231I: The Security component's FFDC
         Diagnostic Module com.ibm.ws.security.core.Secur
red successfully: true.
Trace: 2005/05/06 17:27:40.892 01 t=8E96E0 c=UNK key=P8 (0000000A)
  Description: Log Boss/390 Error
  from filename: ./bborjtr.cpp
  at line: 932
  error message: BBOO0222I: SECJ0231I: The Security component's FFDC
         Diagnostic Module com.ibm.ws.security.core.Securit
d successfully: true.
Trace: 2005/05/06 17:27:41.054 01 t=8E96E0 c=UNK key=P8 (13007002)
  ThreadId: 0000000a
  FunctionName: com.ibm.ws.security.audit.AuditServiceImpl
  SourceId: com.ibm.ws.security.audit.AuditServiceImpl
  Category: AUDIT
  ExtendedMessage: BBOO0222I: SECJ6004I: Security Auditing is disabled.
Trace: 2005/05/06 17:27:41.282 01 t=8E96E0 c=UNK key=P8 (13007002)
  ThreadId: 0000000a
  FunctionName: com.ibm.ws.security.core.distSecurityComponentImpl
  SourceId: com.ibm.ws.security.core.distSecurityComponentImpl
  Category: INFO
  ExtendedMessage: BBOO0222I: SECJ0309I: Java 2 Security is disabled.
Trace: 2005/05/06 17:27:41.282 01 t=8E96E0 c=UNK key=P8 (0000000A)
  Description: Log Boss/390 Error
  from filename: ./bborjtr.cpp
  at line: 932
  error message: BBOO0222I: SECJ0309I: Java 2 Security is disabled.
Trace: 2005/05/06 17:27:42.239 01 t=8E96E0 c=UNK key=P8 (13007002)
  ThreadId: 0000000a
  FunctionName: com.ibm.ws.security.auth.login.Configuration
  SourceId: com.ibm.ws.security.auth.login.Configuration
  Category: AUDIT
  ExtendedMessage: BBOO0222I: SECJ0215I: Successfully set JAAS login
         provider configuration class to com.ibm.ws.securit
Configuration.
Trace: 2005/05/06 17:27:42.253 01 t=8E96E0 c=UNK key=P8 (13007002)
  ThreadId: 0000000a
  FunctionName: com.ibm.ws.security.core.distSecurityComponentImpl
  SourceId: com.ibm.ws.security.core.distSecurityComponentImpl
  Category: INFO
  ExtendedMessage: BBOO0222I: SECJ0212I: WCCM JAAS configuration information
         successfully pushed to login provider clas
Trace: 2005/05/06 17:27:42.254 01 t=8E96E0 c=UNK key=P8 (0000000A)
  Description: Log Boss/390 Error
  from filename: ./bborjtr.cpp
  at line: 932
  error message: BBOO0222I: SECJ0212I: WCCM JAAS configuration information
```

```
          successfully pushed to login provider class.
Trace: 2005/05/06 17:27:42.306 01 t=8E96E0 c=UNK key=P8 (13007002)
  ThreadId: 0000000a
  FunctionName: com.ibm.ws.security.core.distSecurityComponentImpl
  SourceId: com.ibm.ws.security.core.distSecurityComponentImpl
  Category: INFO
  ExtendedMessage: BBOO0222I: SECJ0240I: Security service initialization
         completed successfully
Trace: 2005/05/06 17:27:42.306 01 t=8E96E0 c=UNK key=P8 (0000000A)
  Description: Log Boss/390 Error
  from filename: ./bborjtr.cpp
  at line: 932
  error message: BBOO0222I: SECJ0240I: Security service initialization
          completed successfully
Trace: 2005/05/06 17:27:42.952 01 t=8E96E0 c=UNK key=P8 (13007002)
  ThreadId: 0000000a
  FunctionName: com.ibm.ws.objectpool.ObjectPoolService
  SourceId: com.ibm.ws.objectpool.ObjectPoolService
  Category: INFO
  ExtendedMessage: BBOO0222I: OBPL0007I: Object Pool Manager service
         is disabled.
Trace: 2005/05/06 17:27:53.512 01 t=8E96E0 c=UNK key=P8 (13007002)
  ThreadId: 0000000a
  FunctionName: com.ibm.ws.security.registry.UserRegistryImpl
  SourceId: com.ibm.ws.security.registry.UserRegistryImpl
  Category: AUDIT
  ExtendedMessage: BBOO0222I: SECJ0136I: Custom
         Registry:com.ibm.ws.security.registry.zOS.SAFRegistryImpl
         has been init
Trace: 2005/05/06 17:27:55.229 01 t=8E96E0 c=UNK key=P8 (13007002)
  ThreadId: 0000000a
  FunctionName: com.ibm.ws.security.role.PluggableAuthorizationTableProxy
  SourceId: com.ibm.ws.security.role.PluggableAuthorizationTableProxy
  Category: AUDIT
  ExtendedMessage: BBOO0222I: SECJ0157I: Loaded Vendor
         AuthorizationTable: com.ibm.ws.security.core.SAFAuthorizationTab
Trace: 2005/05/06 17:27:56.481 01 t=8E96E0 c=UNK key=P8 (13007002)
  ThreadId: 0000000a
  FunctionName: com.ibm.ws.security.core.distSecurityComponentImpl
  SourceId: com.ibm.ws.security.core.distSecurityComponentImpl
  Category: INFO
  ExtendedMessage: BBOO0222I: SECJ0243I: Security service started successfully
Trace: 2005/05/06 17:27:56.481 01 t=8E96E0 c=UNK key=P8 (0000000A)
  Description: Log Boss/390 Error
  from filename: ./bborjtr.cpp
  at line: 932
  error message: BBOO0222I: SECJ0243I: Security service started successfully
Trace: 2005/05/06 17:27:56.482 01 t=8E96E0 c=UNK key=P8 (13007002)
  ThreadId: 0000000a
  FunctionName: com.ibm.ws.security.core.distSecurityComponentImpl
  SourceId: com.ibm.ws.security.core.distSecurityComponentImpl
  Category: INFO
  ExtendedMessage: BBOO0222I: SECJ0210I: Security enabled true
Trace: 2005/05/06 17:27:56.483 01 t=8E96E0 c=UNK key=P8 (0000000A)
  Description: Log Boss/390 Error
  from filename: ./bborjtr.cpp
  at line: 932
  error message: BBOO0222I: SECJ0210I: Security enabled true
```

Is there a stack trace or exception printed in the system log file?

A single stack trace tells a lot about the problem. What code initiated the code that failed? What is the failing component? Which class did the failure actually come from? Sometimes the stack trace is all that is needed to solve the problem and it can pinpoint the root cause. Other times, it can only give us a clue, and can actually be misleading. When support analyzes a stack trace, they can request additional trace if it is not clear what the problem is. If it seems to be security-related and the solution cannot be determined from the stack trace or problem description, you are asked to gather the following trace specification: SASRas=all=enabled:com.ibm.ws.security.*=all=enabled from all processes involved.

Is this a distributed security problem or a local security problem?

- If the problem is local, that is the code involved does not make a remote method invocation, then troubleshooting is isolated to a single process. It is important to know when a problem is

local versus distributed because the behavior of the object request broker (ORB), among other components, is different between the two. When a remote method invocation takes place, an entirely different security code path is entered.

- V6.0.x When you know that the problem involves two or more servers, the techniques of troubleshooting change. You need to trace all the servers involved simultaneously so that the trace shows the client and server sides of the problem. Make sure the timestamps on all machines match as closely as possible so that you can find the request and reply pair from two different processes. Enable both Secure Authentication Services (SAS) or z/SAS and Security trace using the trace specification: SASRas=all=enabled:com.ibm.ws.security.*=all=enabled.

  For more information on enabling trace, see Enabling trace.

  For more information on enabling trace, see Working with Trace.

Is the problem related to authentication or authorization?

Most security problems fall under one of these two categories. Authentication is the process of determining who the caller is. Authorization is the process of validating that the caller has the proper authority to invoke the requested method. When authentication fails, typically this failure is related to either the authentication protocol, authentication mechanism or user registry. When authorization fails, this is usually related to the application bindings from assembly and deployment and to the caller's identity who is accessing the method and the roles that are required by the method.

Is this a Web or EJB request?

Web requests have a completely different code path than Enterprise JavaBeans (EJB) requests. Different security features exist for Web requests than for EJB requests, requiring a completely different body of knowledge to resolve. For example, when using the Lightweight Third-Party Authentication (LTPA) authentication mechanism, the single sign-on feature (SSO) is available for Web requests but not for EJB requests. Web requests involve HTTP header information that is not required by EJB requests due to the protocol differences. Also, the Web container or servlet engine is involved in the entire process. Any of these components can be involved in the problem and all require consideration during troubleshooting, based on the type of request and where the failure occurs.

Secure EJB requests heavily involve the ORB and Naming components since they flow over the RMI/IIOP protocol. In addition, when Workload Manager (WLM) is enabled, other behavior changes in the code can be observed. All of these components interact closely for security to work properly in this environment. At times, trace in any or all of these components might be necessary to troubleshoot problems in this area.

V6.0.x The trace specification to begin with is SASRas=all=enabled:com.ibm.ws.security.*=all=enabled. ORB trace is also very beneficial when the SAS/Security trace does not seem to pinpoint the problem.

Does the problem seem to be related to the Secure Sockets Layer (SSL)?

SSL is a totally distinct separate layer of security. Troubleshooting SSL problems is usually separate from troubleshooting authentication and authorization problems, and you have many considerations. Usually, SSL problems are first-time setup problems because the configuration can be difficult. Each client must contain the signer certificate of the server. During mutual authentication, each server must contain the client's signer certificate. Also, there can be protocol differences (SSLv3 vs. Transport Layer Security (TLS)), and listener port problems related to stale Interoperable Object References (IORs), that is IORs from a server, that reflect the port prior to the server restarting.

For SSL problems, sometimes you get a request for an SSL trace to determine what is happening with the SSL handshake. The SSL handshake is the process that occurs when a client opens a socket to a server. If anything goes wrong with the key exchange, cipher exchange, and so on, the handshake fails and the socket is not valid. Tracing JSSE (the SSL implementation that is used in WebSphere Application Server) involves the following steps:

- Set the following system property on the client and server processes: -Djavax.net.debug=true. For the server, add the system property to the generic JVM arguments property of the JVM settings page. For more information on this task, refer to Java virtual machine settings section of the *Administering applications and their environment* PDF book.
- Turn on ORB trace as well.
- Recreate the problem.

  The `SystemOut.log` of both processes contain the JSSE trace. You can find trace similar to the following example:

```
SSLConnection: install <com.ibm.sslite.e@3ae78375>
>> handleHandshakeV2 <com.ibm.sslite.e@3ae78375>
>> handshakeV2 type = 1
>> clientHello: SSLv2.
SSL client version: 3.0
...
...
...
JSSEContext: handleSession[Socket[addr=null,port=0,localport=0]]

<< sendServerHello.
SSL version: 3.0
SSL_RSA_WITH_RC4_128_MD5
HelloRandom
...
...
...
<< sendCertificate.
<< sendServerHelloDone.
>> handleData <com.ibm.sslite.e@3ae78375>
>> handleHandshake <com.ibm.sslite.e@3ae78375>
>> handshakeV3 type = 16

>> clientKeyExchange.
>> handleData <com.ibm.sslite.e@3ae78375>
>> handleChangeCipherSpec <com.ibm.sslite.e@3ae78375>
>> handleData <com.ibm.sslite.e@3ae78375>
>> handleHandshake <com.ibm.sslite.e@3ae78375>
>> handshakeV3 type = 20
>> finished.
<< sendChangeCipherSpec.
<< sendFinished.
```

Trace security

The classes that implement WebSphere Application Server security are:
- com.ibm.ws.security.*
- com.ibm.websphere.security.*
- com.ibm.WebSphereSecurityImpl.*
- SASRas
- com.ibm.ws.wim.* for tracing with a Virtual Member Manager (VMM) repository

To view detailed information on the run time behavior of security, enable trace on the following components and review the output:
- com.ibm.ws.security.*=all=enabled:com.ibm.WebSphereSecurityImpl.*= all=enabled:com.ibm.websphere.security.*=all=enabled. This trace statement collects the trace for the security runtime.
- com.ibm.ws.console.security.*=all=enabled. This trace statement collects the trace for the security center administrative console.
- **V6.0.x** SASRas=all=enabled. This trace statement collects the trace for SAS (low-level authentication logic).
- com.ibm.ws.wim.*=all=enabled:com.ibm.websphere.wim.*=all=enabled. This trace statement collects the trace for VMM.

V6.0.x

Fine tuning Security traces:

If a subset of packages need to be traced, specify a trace specification more detailed than com.ibm.ws.security.*=all=enabled. For example, to trace just dynamic policy code, you can specify com.ibm.ws.security.policy.*=all=enabled. To disable dynamic policy trace, you can specify com.ibm.ws.security.policy.*=all=disabled.

Configuring CSIv2, or SAS Trace Settings

Situations arise where reviewing trace for the CSIv2 or SAS authentication protocols can assist in troubleshooting difficult problems. This section describes how to enable to CSIv2 and SAS trace.

Enabling Client-Side CSIv2 and SAS Trace

To enable CSIv2 and SAS trace on a pure client, the following steps need to be taken:

- Edit the file `TraceSettings.properties` in the `/WebSphere/AppServer/properties` directory. For example, edit *profile_root*`/properties/TraceSettings.properties`.
- In this file, change traceFileName= to point to the path in which you want the output file created. For example, traceFileName=*profile_root*`/logs/sas_client`.
- n this file, add the trace specification string: SASRas=all=enabled. Any additional trace strings can be added on separate lines.
- Point to this file from within your client application. On the Java command line where you launch the client, add the following system property: -DtraceSettingsFile=TraceSettings.properties.

  Note: Do not give the fully qualified path to the `TraceSettings.properties` file. Make sure that the `TraceSettings.properties` file is in your class path.

Enabling Server-Side CSIv2 and SAS Trace

To enable SAS trace in an application server, complete the following:

- Add the trace specification, SASRas=all=enabled, to the `server.xml` file or add it to the Trace settings within the WebConsole GUI.
- Typically it is best to also trace the authorization security runtime in addition to the authentication protocol runtime. To do this, use the following two trace specifications in combination: SASRas=all=enabled:com.ibm.ws.security.*=all=enabled.
- When troubleshooting a connection type problem, it is beneficial to trace both CSIv2 and SAS or CSIv2 and z/SAS and the ORB. To do this, use the following three trace specifications: SASRas=all=enabled:com.ibm.ws.security.*=all=enabled:ORBRas=all=enabled.
- In addition to adding these trace specifications, for ORB trace there are a couple of system properties that also need to be set. Go to the ORB settings in the GUI and add the following two properties: com.ibm.CORBA.Debug=true and com.ibm.CORBA.CommTrace=true.

CSIv2 CORBA minor codes

Whenever exceptions occur within the security code on either the client or server, the eventual exception becomes a Common Object Request Broker Architecture (CORBA) exception. Any exception that occurs gets embedded in a CORBA exception because the CORBA architecture is used by the security service for its own inter-process communication. CORBA exceptions are generic and indicate a problem in communication between two components. CORBA minor codes are more specific and indicate the underlying reason that a component could not complete a request.

The following shows the CORBA minor codes that a client can expect to receive after running a security-related request such as authentication. It also includes the CORBA exception type that the minor code appears in.

The following exception shows an example of a CORBA exception where the minor code is 49424300 and indicates Authentication Failure. Typically, a descriptive message is also included in the exception to assist in troubleshooting the problem. Here, the detailed message is: ″Exception caught invoking

authenticateBasicAuthData from SecurityServer for user jdoe. Reason: com.ibm.WebSphereSecurity.AuthenticationFailedException″ which indicates that the authentication failed for user `jdoe`.

The completed field in the exception indicates whether the method was completed or not. In the case of a NO_PERMISSION, never invoke the message; therefore it is always `completed:No`. Other exceptions that are caught on the server side can have a completed status of ″Maybe″ or ″Yes″.

```
org.omg.CORBA.NO_PERMISSION: Caught WSSecurityContextException in
WSSecurityContext.acceptSecContext(),
reason: Major Code[0] Minor Code[0] Message[Exception caught invoking
authenticateBasicAuthData from SecurityServer for user jdoe.  Reason:
com.ibm.WebSphereSecurity.AuthenticationFailedException]  minor code: 49424300
completed: No

at com.ibm.ISecurityLocalObjectBaseL13Impl.PrincipalAuthFailReason.
map_auth_fail_to_minor_code(PrincipalAuthFailReason.java:83)
        at com.ibm.ISecurityLocalObjectBaseL13Impl.CSIServerRI.receive_request
               (CSIServerRI.java:1569)
        at com.ibm.rmi.pi.InterceptorManager.iterateReceiveRequest
              (InterceptorManager.java:739)
        at com.ibm.CORBA.iiop.ServerDelegate.dispatch(ServerDelegate.java:398)
        at com.ibm.rmi.iiop.ORB.process(ORB.java:313)
        at com.ibm.CORBA.iiop.ORB.process(ORB.java:1581)
        at com.ibm.rmi.iiop.GIOPConnection.doWork(GIOPConnection.java:1827)
        at com.ibm.rmi.iiop.WorkUnitImpl.doWork(WorkUnitImpl.java:81)
        at com.ibm.ejs.oa.pool.PooledThread.run(ThreadPool.java:91)
        at com.ibm.ws.util.CachedThread.run(ThreadPool.java:149)
```

The following table shows the CORBA minor codes which a client can expect to receive after running a security-related request such as authentication. It also includes the CORBA exception type that the minor code would appear in.

Table 3.

Minor code name	Minor code value (in hex)	Exception type (all in the package of org.omg.CORBA .*)	Minor code description	Retry performed (when authenticationRetryEnabled = true)
AuthenticationFailed	49424300	NO_PERMISSION	This code is a generic authentication failed error. It does not give any details about whether or not the user ID or password is valid. Some user registries can choose to use this type of error code, others can choose to use the next three types that are more specific.	Yes
InvalidUserid	49424301	NO_PERMISSION	This code occurs when the registry returns bad user ID.	Yes
InvalidPassword	49424302	NO_PERMISSION	This code occurs when the registry returns a bad password.	Yes
InvalidSecurityCredentials	49424303	NO_PERMISSION	This is a generic error indicating that the credentials are bad for some reason. It might be that the right attributes are not set.	Yes, if client has BasicAuth credential (token based credential was rejected in the first place).
InvalidRealm	49424304	NO_PERMISSION	This code occurs when the REALM in the token received from the client does not match the server's current realm.	No
ValidationFailed	49424305	NO_PERMISSION	A validation failure occurs when a token is sent from the client or server to a target server but the token format or the expiration is not valid.	Yes, if client has BasicAuth credential (token based credential was rejected in the first place).

Table 3. (continued)

Minor code name	Minor code value (in hex)	Exception type (all in the package of org.omg.CORBA .*)	Minor code description	Retry performed (when authenticationRetryEnabled = true)
CredentialTokenExpired	49424306	NO_PERMISSION	This code is more specific about why the validation failed. In this case, the token has an absolute lifetime and the lifetime has expired. Therefore, it is no longer a valid token and cannot be used.	Yes, if client has BasicAuth credential (token based credential was rejected in the first place).
InvalidCredentialToken	49424307	NO_PERMISSION	This is more specific about why the validation failed. In this case, the token cannot be decrypted or the data within the token is not readable.	Yes, if client has BasicAuth credential (token based credential was rejected in the first place).
SessionDoesNotExist	49424308	NO_PERMISSION	This indicates that the CSIv2 session does not exist on the server. Typically, a retry occurs automatically and successfully creates a new session.	Yes
SessionConflictingEvidence	49424309	NO_PERMISSION	This indicates that a session already exists on the server that matches the context_id sent over by the client. However, the information provided by the client for this EstablishContext message is different from the information originally provided to establish the session.	Yes
SessionRejected	4942430A	NO_PERMISSION	This indicates that the session referenced by the client has been previously rejected by the server.	Yes
SecurityServerNotAvailable	4942430B	NO_PERMISSION	This error occurs when the server cannot contact the local or remote security server in order to authenticate or validate.	No
InvalidIdentityToken	4942430C	NO_PERMISSION	This error indicates that identity cannot be obtained from the identity token when Identity Assertion is enabled.	No
IdentityServerNotTrusted	4942430D	NO_PERMISSION	This indicates that the server ID of the sending server is not on the target server's trusted principal list.	No
InvalidMessage	4942430E	NO_PERMISSION	This indicates that the CSIv2 message format is not valid for the receiving server.	No
AuthenticationNotSupported	49421090	NO_PERMISSION	This error occurs when a mechanism does not support authentication (very rare).	No
InvalidSecurityMechanism	49421091	NO_PERMISSION	This is used to indicate that the specified security mechanism is not known.	No
CredentialNotAvailable	49421092	NO_PERMISSION	This indicates a credential is not available when it is required.	No
SecurityMechanismNotSupported	49421093	NO_PERMISSION	This error occurs when a security mechanism that is specified in the CSIv2 token is not implemented on the server.	No

Table 3. (continued)

Minor code name	Minor code value (in hex)	Exception type (all in the package of org.omg.CORBA .*)	Minor code description	Retry performed (when authenticationRetryEnabled = true)
ValidationNotSupported	49421094	NO_PERMISSION	This error occurs when a mechanism does not support validation, such as LocalOS. This error does not occur since the LocalOS credential is not a forwardable credential, therefore, validation never needs to be called on this credential.	No
CredentialTokenNotSet	49421095	NO_PERMISSION	This is used to indicate that the token inside the credential is null.	No
ServerConnectionFailed	494210A0	COMM_FAILURE	This error is used when a connection attempt fails.	Yes (via ORB retry)
CorbaSystemException	494210B0	INTERNAL	This code is a generic CORBA specific exception in system code.	No
JavaException	494210B1	INTERNAL	This is a generic error that indicated that an unexpected Java exception occurred.	No
ValueIsNull	494210B2	INTERNAL	This code is used to indicate that a value or parameter that passed in is null.	No
EffectivePolicyNotPresent	494210B3	INTERNAL	This indicates that an effective policy object for CSIv2 is not present. This object is used to determine what security configuration features are specified.	No
NullPointerException	494210B4	INTERNAL	This code is used to indicate that a NullPointerException is caught in the runtime.	No
ErrorGettingClassInstance	494210B5	INTERNAL	This indicates a problem loading a class dynamically.	No
MalFormedParameters	494210B6	INTERNAL	This indicates parameters are not valid.	No
DuplicateSecurityAttributeType	494210B7	INTERNAL	This indicates a duplicate credential attribute that is specified during the set_attributes operation.	No
MethodNotImplemented	494210C0	NO_IMPLEMENT	This indicates that a method invoked is not implemented.	No
GSSFormatError	494210C5	BAD_PARAM	This code indicates that a Generic Security Services (GSS) encoding or decoding routine has created an exception.	No
TagComponentFormatError	494210C6	BAD_PARAM	This code indicates that a tag component cannot be read properly.	No
InvalidSecurityAttributeType	494210C7	BAD_PARAM	This code indicates an attribute type specified during the set_attributes operation is not a valid type.	No
SecurityConfigError	494210CA	INITIALIZE	This code indicates a problem exists between the client and server configuration.	No

For current information available from IBM Support on known problems and their resolution, see the IBM Support page.

IBM Support has documents that can save you time gathering information needed to resolve this problem. Before opening a PMR, see the IBM Support page.

Errors when trying to configure or enable security

Use this information to troubleshoot problems with configuring or enabling security.

What kind of error are you seeing?

- “″LTPA password not set. validation failed″ message displayed as error in the administrative console after saving administrative or application security settings ”
- “Errors when trying to configure or enable security”
- “WebSphere Application Server Version 6 is not working correctly with Enterprise Workload Manager (EWLM)”
- If you successfully configured security, but are now having problems accessing Web resources or the administrative console, refer to Errors or access problems after enabling security.
- “NMSV0610I: A NamingException is being thrown from a javax.naming.Context implementation” on page 335

For general tips on diagnosing and resolving security-related problems, see the topic Troubleshooting the security component.

IBM Support has documents and tools that can save you time gathering information needed to resolve problems as described in “Troubleshooting help from IBM” on page 126. Before opening a problem report, see the Support page:

- http://www.ibm.com/servers/eserver/support/iseries/software/v5r3/index.html

″LTPA password not set. validation failed″ message displayed as error in the administrative console after saving administrative or application security settings

This error can be caused if, when configuring WebSphere Application Server security, LTPA is selected as the authentication mechanism and the LTPA password field is not set. To resolve this problem:

- Select **Security > Secure administration, applications and infrastructure > Authentication mechanisms and expiration** .
- Complete the password and confirm password fields.
- Click **OK**.
- Try setting administrative or application security again.

WebSphere Application Server Version 6 is not working correctly with Enterprise Workload Manager (EWLM)

To use WebSphere Application Server Version 6 with EWLM, you must manually update the WebSphere Application Serve server.policy files. For example:

```
grant codeBase "file:/<EWLM_Install_Home>/classes/ARM/arm4.jar" {
 permission java.security.AllPermission;
};
```

Otherwise, you might encounter a Java 2 security exception for violating the Java 2 security permission.

For more information on configuring server.policy files, refer to the server.policy file permissions section in the *Developing and deploying applications* PDF book.

For current information available from IBM Support on known problems and their resolution, see the IBM Support page.

IBM Support has documents that can save you time gathering information needed to resolve this problem. Before opening a PMR, see the IBM Support page.

NMSV0610I: A NamingException is being thrown from a javax.naming.Context implementation

If you use CSIv2 inbound authentication, basic authentication is required, and Java clients running with com.ibm.CORBA.validateBasicAuth=true might fail with the following exception:

```
If you use CSIv2 inbound authentication, basic authentication is required, and Java™
clients running with com.ibm.CORBA.validateBasicAuth=true might fail with the
following exception:

NMSV0610I: A NamingException is being thrown from a javax.naming.Context
           implementation. Details follow:

Context implementation: com.ibm.ws.naming.jndicos.CNContextImpl
Context method: lookupExt
Context name: TestaburgerNode01Cell/nodes/TestaburgerNode01/servers/server1
Target name: SecurityServer
Other data: ""
Exception stack trace: javax.naming.NoPermissionException: NO_PERMISSION
exception caught. Root exception is org.omg.CORBA.NO_PERMISSION:
vmcid: 0x49421000 minor code: 92 completed: No
...
SECJ0395E: Could not locate the SecurityServer at host/port:9.42.72.27/9100
to validate the userid and password entered. You may need to specify valid
securityServerHost/Port in (WAS_INSTALL_ROOT)/properties/sas.client.props file.
```

To fix this problem, modify the `com.ibm.CORBA.validateBasicAuth=false` property in the `clients.sas.clients.props` file and then run the client.

Errors after enabling security

Use this information if you are experiencing errors after security is enabled.

What kind of error are you seeing?

- Authentication error accessing a Web page
- Authorization error accessing a Web page
- Error Message: `CWSCJ0314E: Current Java 2 security policy reported a potential violation`
- `CWMSG0508E: The JMS Server security service was unable to authenticate user ID:` error displayed in SystemOut.log when starting an application server
- Error Message: `CWSCJ0237E: One or more vital LTPAServerObject configuration attributes are null or not available` after enabling security and starting the application server
- An AccessControlException is reported in the SystemOut.log
- Error Message: `CWSCJ0336E: Authentication failed for user {0} because of the following exception {1}`

For general tips on diagnosing and resolving security-related problems, see the topic Troubleshooting the security component.

IBM Support has documents and tools that can save you time gathering information needed to resolve problems as described in “Troubleshooting help from IBM” on page 126. Before opening a problem report, see the Support page:

- http://www.ibm.com/servers/eserver/support/iseries/software/v5r3/index.html

Authentication error accessing a Web page

Possible causes for authentication errors include:

- **Incorrect user name or passwords.** Check the user name and password and make sure that they are correct.

- **Security configuration error : User registry type is not set correctly.** Check the user registry property in administrative security settings in the administrative console. Verify that the user registry property is the intended user registry.
- **Internal program error.** If the client application is a Java standalone program, this program might not gather or send credential information correctly.

If the user registry configuration, user ID, and password appear correct, use the WebSphere Application Server trace to determine the cause of the problem. To enable security trace, use the com.ibm.ws.security.*=all=enabled trace specification.

Authorization error accessing a Web page

If a user who is supposed to have access to a resource does not, a configuration step is probably missing. For more information on configuring access to resources, review the chapter Authorizing access to administrative roles in the *Securing applications and their environment* PDF book.

Specifically:
- Check the required roles for the accessed Web resource.
- Check the authorization table to make sure that the user, or the groups to which the user belongs, is assigned to one of the required roles.
- View required roles for the Web resource in the deployment descriptor of the Web resource.
- View the authorization table for the application that contains the Web resource, using the administrative console.
- Test with a user who is granted the required roles, to see if the user can access the problem resources.
- If the user is required to have one or more of the required roles, use the administrative console to assign that user to required roles, stop, and restart the application.

If the user is granted required roles, but still fails to access the secured resources, enable security trace, using com.ibm.ws.security.*=all=enabled as the trace specification. Collect trace information for further resolution.

Error Message: `CWSCJ0314E: Current Java 2 security policy reported a potential violation on server`

If you find errors on your server similar to:

```
Error Message: CWSCJ0314E: Current Java 2 Security policy reported a potential violation of
Java 2 Security Permission. Please refer to Problem Determination Guide for further information.
{0}Permission/:{1}Code/:{2}{3}Stack Trace/:{4}Code Base Location/:{5}
```

The Java security manager checkPermission method has reported a SecurityException exception .

The reported exception might be critical to the secure system. Turn on security trace to determine the potential code that might have violated the security policy. Once the violating code is determined, verify if the attempted operation is permitted with respect to Java 2 Security, by examining all applicable Java 2 security policy files and the application code.

A more detailed report is enabled by either configuring RAS trace into debug mode, or specifying a Java property.
- Check the trace enabling section for instructions on how to configure Reliability Availability Serviceability (RAS) trace into debug mode, or
- Specify the following property in the **Application Servers** > *server_name* > **ProcessDefinition** > **Java Virtual Machine** panel from the administrative console in the **Generic JVM arguments** panel:
  - Add the **java.security.debug** run-time flag
  - Valid values:

access
Print all debug information including required permission, code, stack, and code base location.
stack Print debug information including required permission, code, and stack.
failure Print debug information including required permission, and code.

For a review of Java security policies, see the Java 2 Security documentation at http://java.sun.com/j2se/1.3/docs/guide/security/index.html.

Tip: If the application is running with a Java Mail application programming interface (API), this message might be benign. You can update the *installed Enterprise Application root*/META-INF/was.policy file to grant the following permissions to the application:
- permission java.io.FilePermission ″${user.home}${/}.mailcap″, ″read″;
- permission java.io.FilePermission ″${user.home}${/}.mime.types″, ″read″;
- permission java.io.FilePermission ″${java.home}${/}lib${/}mailcap″, ″read″;
- permission java.io.FilePermission ″${java.home}${/}lib${/}mime.types″, ″read″;

Error message: CWMSG0508E: The JMS Server security service was unable to authenticate user ID:″ error displayed in SystemOut.log when starting an application server

This error can result from installing the Java Message Service (JMS) API sample and then enabling security. You can follow the instructions in the Configure and Run page of the corresponding JMS sample documentation to configure the sample to work with WebSphere Application Server security.

You can verify the installation of the message-driven bean sample by launching the installation program, selecting **Custom**, and browsing the components which are already installed in the Select the features you like to install panel. The JMS sample is shown as **Message-Driven Bean Sample**, under Embedded Messaging.

You can also verify this installation by using the administrative console to open the properties of the application server that contains the samples. Select **MDBSamples** and click uninstall.

Error message: CWSCJ0237E: One or more vital LTPAServerObject configuration attributes are null or not available after enabling security and starting the application server.

This error message can result from selecting Lightweight Third Party Authentication (LTPA) as the authentication mechanism, but not generating the LTPA keys. The LTPA keys encrypt the LTPA token.

To resolve this problem:
1. Click **Security > Secure administration, applications and infrastructure> Authentication > Authentication mechanisms and expiration> LTPA**
2. Enter a password, which can be anything.
3. Enter the same password in **Confirm Password**.
4. Click **Apply**.
5. Click **Generate Keys**.
6. Click **Save**.

The AccessControlException exception, is reported in the SystemOut.log

The problem is related to the Java 2 security feature of WebSphere Application Server, the API-level security framework that is implemented in WebSphere Application Server. An exception similar to the following example displays. The error message and number can vary.

```
CWSRV0020E: [Servlet Error]-[validator]: Failed to load servlet:
java.security.AccessControlException: access denied
(java.io.FilePermission
app_server_root/systemApps/isclite.ear/isclite.war/WEB-INF/validation.xml read)
```

For an explanation of Java 2 security, how and why to enable or disable it, how it relates to policy files, and how to edit policy files, see the Java 2 security topic in the *Securing applications and their environment* PDF book.The topic explains that Java 2 security is not only used by this product, but developers can also implement it for their business applications. Administrators might need to involve developers, if this exception is created when a client tries to access a resource that is hosted by WebSphere Application Server.

Possible causes of these errors include:
- Syntax errors in a policy file.
- Syntax errors in permission specifications in the ra.xml file that is bundled in a .rar file. This case applies to resource adapters that support connector access to CICS or other resources.
- An application is missing the specified permission in a policy file, or in permission specifications in ra.xml file bundled in a .rar file.
- The class path is not set correctly, preventing the permissions for the resource.xml file for Service Provider Programming Interface (SPI) from being correctly created.
- A library called by an application, or the application, is missing a doPrivileged block to support access to a resource.
- Permission is specified in the wrong policy file.

To resolve these problems:
- Check all of the related policy files to verify that the permission shown in the exception, for example java.io.FilePermission, is specified.
- Look for a related ParserException exception in the SystemOut.log file which reports the details of the syntax error. For example:

  ```
  CWSCJ0189E: Caught ParserException while creating template for application policy

  profile_root/config/cells/cell_name/nodes/node_name/app.policy
  ```

  Where:
  - *cell_name* represents the name of your cell.
  - *profile_name* represents the name of your profile.
  - *node_name* represents the name of your node.

  The exception is com.ibm.ws.security.util.ParserException: line 18: expected ';', found 'grant'
- Look for a message similar to: `CWSCJ0325W: The permission permission specified in the policy file is unresolved.`
- Check the call stack to determine which method does not have the permission. Identify the class path of this method. If it is hard to identify the method, enable the Java2 security Report.
  - Configuring RAS trace by specifying `com.ibm.ws.security.core.*=all=enabled`, or specifying a Java **property.java.security.debug** property. Valid values for the **java.security.debug** property are:

    access
    Print all debug information including: required permission, code, stack, and code base location.

    stack Print debug information including: required permission, code, and stack.

    failure Print debug information including: required permission and code.
  - The report shows:

    Permission
    The missing permission.

    Code Which method has the problem.

    Stack Trace
    Where the access violation occurred.

    CodeBaseLocation
    The detail of each stack frame.

Usually, permission and code are enough to identify the problem. The following example illustrates a report:

```
Permission:
profile_root/logs/server1/SystemOut_02.08.20_11.19.53.log :
access denied (java.io.FilePermission
profile_root/logs/server1/SystemOut_02.08.20_11.19.53.log delete)

Code:
 com.ibm.ejs.ras.RasTestHelper$7  in
{file:profile_root/installedApps/app1/JrasFVTApp.ear/RasLib.jar
}

Stack Trace:

java.security.AccessControlException: access denied (java.io.FilePermission
profile_root/logs/server1/SystemOut_02.08.20_11.19.53.log delete
)
        at java.security.AccessControlContext.checkPermission
                           (AccessControlContext.java(Compiled Code))
        at java.security.AccessController.checkPermission
                           (AccessController.java(Compiled Code))
        at java.lang.SecurityManager.checkPermission
                           (SecurityManager.java(Compiled Code))
                            .
Code Base Location:

com.ibm.ws.security.core.SecurityManager :
file:app_server_root/plugins/com.ibm.ws.runtime_6.1.0.jar

  ClassLoader: com.ibm.ws.bootstrap.ExtClassLoader
  Permissions granted to CodeSource
(file:app_server_root/plugins/com.ibm.ws.runtime_6.1.0.jar <no certificates>
  {
    (java.util.PropertyPermission java.vendor read);
    (java.util.PropertyPermission java.specification.version read);
    (java.util.PropertyPermission line.separator read);
    (java.util.PropertyPermission java.class.version read);
    (java.util.PropertyPermission java.specification.name read);
    (java.util.PropertyPermission java.vendor.url read);
    (java.util.PropertyPermission java.vm.version read);
    (java.util.PropertyPermission os.name read);
    (java.util.PropertyPermission os.arch read);
   }
   ( This list continues.)
Permission:
profile_root/logs/server1/SystemOut_02.08.20_11.19.53.log :
access denied (java.io.FilePermission
profile_root/logs/server1/SystemOut_02.08.20_11.19.53.log delete)

Code:
 com.ibm.ejs.ras.RasTestHelper$7  in
{file:profile_root/installedApps/app1/JrasFVTApp.ear/RasLib.jar}

Stack Trace:

java.security.AccessControlException: access denied (java.io.FilePermission

profile_root/logs/server1/SystemOut_02.08.20_11.19.53.log delete)
        at java.security.AccessControlContext.checkPermission
                           (AccessControlContext.java(Compiled Code))
        at java.security.AccessController.checkPermission
                           (AccessController.java(Compiled Code))
        at java.lang.SecurityManager.checkPermission
                           (SecurityManager.java(Compiled Code))
                            .
Code Base Location:
```

```
com.ibm.ws.security.core.SecurityManager :

file:app_server_root/plugins/com.ibm.ws.runtime_6.1.0.jar
  ClassLoader: com.ibm.ws.bootstrap.ExtClassLoader
  Permissions granted to CodeSource
(file:app_server_root/plugins/com.ibm.ws.runtime_6.1.0.jar <no certificates>
  {
    (java.util.PropertyPermission java.vendor read);
    (java.util.PropertyPermission java.specification.version read);
    (java.util.PropertyPermission line.separator read);
    (java.util.PropertyPermission java.class.version read);
    (java.util.PropertyPermission java.specification.name read);
    (java.util.PropertyPermission java.vendor.url read);
    (java.util.PropertyPermission java.vm.version read);
    (java.util.PropertyPermission os.name read);
    (java.util.PropertyPermission os.arch read);
   }
   ( This list continues.)
Permission:
app_server_root/profile1/
logs/server1/SystemOut_02.08.20_11.19.53.log :
access denied (java.io.FilePermission
app_server_root/profile1/
logs/server1/SystemOut_02.08.20_11.19.53.log delete)

Code:
 com.ibm.ejs.ras.RasTestHelper$7  in
{file:app_server_root/profile1/
installedApps/app1/JrasFVTApp.ear/RasLib.jar}

Stack Trace:

java.security.AccessControlException: access denied (java.io.FilePermission

app_server_root/profile1/
logs/server1/SystemOut_02.08.20_11.19.53.log delete)
        at java.security.AccessControlContext.checkPermission
                           (AccessControlContext.java(Compiled Code))
        at java.security.AccessController.checkPermission
                           (AccessController.java(Compiled Code))
        at java.lang.SecurityManager.checkPermission
                           (SecurityManager.java(Compiled Code))
                            .
Code Base Location:

com.ibm.ws.security.core.SecurityManager :

file:app_server_root/plugins/com.ibm.ws.runtime_6.1.0.jar
  ClassLoader: com.ibm.ws.bootstrap.ExtClassLoader
  Permissions granted to CodeSource
(file:app_server_root/plugins/com.ibm.ws.runtime_6.1.0.jar <no certificates>
  {
    (java.util.PropertyPermission java.vendor read);
    (java.util.PropertyPermission java.specification.version read);
    (java.util.PropertyPermission line.separator read);
    (java.util.PropertyPermission java.class.version read);
    (java.util.PropertyPermission java.specification.name read);
    (java.util.PropertyPermission java.vendor.url read);
    (java.util.PropertyPermission java.vm.version read);
    (java.util.PropertyPermission os.name read);
    (java.util.PropertyPermission os.arch read);
   }
   ( This list continues.)
```

Where:

- *app1* represents the name of your application.

  - *app_server_root* represents the installation root directory for WebSphere Application Server.
  - *profile_root* represents the location and name of a particular profile in your system.
  - *profile1* or *profile_name* represents the name of your profile.
  - *server1* or *server_name*represents the name of your application server.
- If the method is SPI, check the resources.xml file to ensure that the class path is correct.
- To confirm that all of the policy files are loaded correctly, or what permission each class path is granted, enable the trace with **com.ibm.ws.security.policy.*=all=enabled**. All loaded permissions are listed in the trace.log file. Search for the app.policy, was.policy and ra.xml files. To check the permission list for a class path, search for **Effective Policy for *classpath***.
- If there are any syntax errors in the policy file or the ra.xml file, correct them with the policy tool. Avoid editing the policy manually, because syntax errors can result. For additional information about using this tool, refer to the section Using PolicyTool to edit policy files in the *Developing and deploying applications* PDF book.
- If a permission is listed as Unresolved, it does not take effect. Verify that the specified permission name is correct.
- If the class path that is specified in the resource.xml file is not correct, correct it.
- If a required permission does not exist in either the policy files or the ra.xml file, examine the application code to see if you need to add this permission. If so, add it to the proper policy file or the ra.xml file.
- If the permission is not granted outside of the specific method that is accessing this resource, modify the code needs to use a doPrivileged block.
- If this permission does exist in a policy file or a ra.xml file and the permission was loaded correctly, but the class path still does not have the permission in its list, the location of the permission might not be correct. Read the Java 2 security chapter in the *Securing applications and their environment* PDF book carefully to determine in which policy file or ra.xml file to specify that permission.

Tip: If the application is running with the Java Mail API, you can update the *installed Enterprise Application root*/META-INF/was.policy file to grant the following permissions to the application:
- permission java.io.FilePermission ″${user.home}${/}.mailcap″, ″read″;
- permission java.io.FilePermission ″${user.home}${/}.mime.types″, ″read″;
- permission java.io.FilePermission ″${java.home}${/}lib${/}mailcap″, ″read″;
- permission java.io.FilePermission ″${java.home}${/}lib${/}mime.types″, ″read″;

Error Message: `CWSCJ0336E: Authentication failed for user {0} because of the following exception {1}`

This error message results if the user ID that is indicated is not found in the Lightweight Directory Access Protocol (LDAP) user registry. To resolve this problem:
1. Verify that your user ID and password are correct.
2. Verify that the user ID exists in the registry.
3. Verify that the base distinguished name (DN) is correct.
4. Verify that the user filter is correct.
5. Verify that the bind DN and the password for the bind DN are correct. If the bind DN and password are not specified, add the missing information and retry.
6. Verify that the host name and LDAP type are correct.

Consult with the administrator of the user registry if the problem persists.

Access problems after enabling security

Use this information if you are experiencing access problems after enabling security.

What kind of error are you seeing?
- I cannot access all or part of the administrative console or use the wsadmin tool after enabling security
- I cannot access a Web page after enabling security
- Authentication error accessing a Web page
- Authorization error accessing a Web page
- The client cannot access an enterprise bean after enabling security

- The client never gets prompted when accessing a secured enterprise bean
- I cannot stop an application server, node manager, or node after enabling security
- AccessControlException is reported in SystemOut.log
- After enabling single sign-on, I cannot log on to the administrative console
- The following exception displays in the SystemOut.log file after I start the server and enable security: ″SECJ0306E: No received or invocation credential exists on the thread.″

For general tips on diagnosing and resolving security-related problems, see the topic Troubleshooting the security component.

If you do not see a problem that resembles yours, or if the information provided does not solve your problem, see “Troubleshooting help from IBM” on page 126.

I cannot access all or part of the administrative console or use the wsadmin tool after enabling security

- If you cannot access the administrative console, or view and update certain objects, look in the SystemOut log of the application server which hosts the administrative console page for a related error message.
- You might not have authorized your ID for administrative tasks. This problem is indicated by errors such as:
  - `[8/2/02 10:36:49:722 CDT] 4365c0d9 RoleBasedAuth A CWSCJ0305A: Role based authorization check failed for security name MyServer/myUserId, accessId MyServer/S-1-5-21-882015564-4266526380-2569651501-1005 while invoking method getProcessType on resource Server and module Server.`
  - Exception message: `″CWWMN0022E: Access denied for the getProcessType operation on Server MBean″`
  - `When running the command: wsadmin -username j2ee -password j2ee: CWWAX7246E: Cannot establish ″SOAP″ connection to host ″BIRKT20″ because of an authentication failure. Ensure that user and password are correct on the command line or in a properties file.`

  To grant an ID administrative authority, from the administrative console, click **System Administration > Console Users** and validate that the ID is a member. If the ID is not a member, add the ID with at least monitor access privileges, for read-only access.
- Verify that the enable_trusted_application flag is set to `true`. To check the enable_trusted_application flag value using the administrative console, click **Security > Secure administration, applications and infrastructure**. Under Additional properties, click **Custom properties > EnableTrustedApplications**.

I cannot access a Web page after enabling security

When secured resources are not accessible, probable causes include:

- Authentication errors - WebSphere Application Server security cannot identify the ID of the person or process. Symptoms of authentication errors include:

  On a Netscape browser:
  - `Authorization failed. Retry?` message is displayed after an attempt to log in.
  - Accepts any number of attempts to retry login and displays `Error 401` message when Cancel is clicked to stop retry.
  - A typical browser message displays: `Error 401: Basic realm='Default Realm'.`

  On an Internet Explorer browser:
  - Login prompt displays again after an attempt to log in.
  - Allows three attempts to retry login.
  - Displays `Error 401` message after three unsuccessful retries.
- Authorization errors - The security function has identified the requesting person or process as not authorized to access the secured resource. Symptoms of authorization errors include:
  - Netscape browser: ″Error 403: AuthorizationFailed″ message is displayed.
  - Internet Explorer:
    - ″You are not authorized to view this page″ message is displayed.

- ″HTTP 403 Forbidden″ error is also displayed.

- SSL errors - WebSphere Application Server security uses Secure Sockets Layer (SSL) technology internally to secure and encrypt its own communication, and incorrect configuration of the internal SSL settings can cause problems. Also you might have enabled SSL encryption for your own Web application or enterprise bean client traffic which, if configured incorrectly, can cause problems regardless of whether WebSphere Application Server security is enabled.
  - SSL-related problems are often indicated by error messages that contain a statement such as: `ERROR: Could not get the initial context or unable to look up the starting context.Exiting.` followed by `javax.net.ssl.SSLHandshakeException`

The client cannot access an enterprise bean after enabling security

If the client access to an enterprise bean fails after security is enabled:

- Review the steps for securing and granting access to resources.
- Browse the server JVM logs for errors relating to enterprise bean access and security. Look up any errors in the message table.

  Errors similar to `Authorization failed for /UNAUTHENTICATED while invoking` *resource* `securityName:/UNAUTHENTICATED;accessId:UNAUTHENTICATED not granted any of the required roles` *roles* indicate that:
  - An unprotected servlet or JavaServer Pages (JSP) file accessed a protected enterprise bean. When an unprotected servlet is accessed, the user is not prompted to log in and the servlet runs as UNAUTHENTICATED. When the servlet makes a call to an enterprise bean that is protected, the servlet fails.

    To resolve this problem, secure the servlet that is accessing the protected enterprise bean. Make sure that the runAs property for the servlet is set to an ID that can access the enterprise bean.
  - An unauthenticated Java client program is accessing an enterprise bean resource that is protected. This situation can happen if the file that is read by the `sas.client.props` properties file that is used by the client program does not have the `securityEnabled` flag set to `true`.

    To resolve this problem, make sure that the `sas.client.props` file on the client side has its securityEnabled flag set to `true`.

  Errors similar to Authorization failed for *valid_user* `while invoking` *resource* `securityName:/username;accessId:xxxxxx not granted any of the required roles` *roles* indicate that a client attempted to access a secured enterprise bean resource, and the supplied user ID is not assigned the required roles for that enterprise bean.
  - Check that the required roles for the enterprise bean resource are accessed. View the required roles for the enterprise bean resource in the deployment descriptor of the Web resource.
  - Check the authorization table and make sure that the user or the group that the user belongs to is assigned one of the required roles. You can view the authorization table for the application that contains the enterprise bean resource using the administrative console.

If `org.omg.CORBA.NO_PERMISSION` exceptions occur when programmatically logging on to access a secured enterprise bean, an authentication exception has occurred on the server. Typically the CORBA exception is triggered by an underlying `com.ibm.WebSphereSecurity.AuthenticationFailedException`. To determine the actual cause of the authentication exception, examine the full trace stack:

1. Begin by viewing the text following `WSSecurityContext.acceptSecContext(), reason:` in the exception. Typically, this text describes the failure without further analysis.
2. If this action does not describe the problem, look up the Common Object Request Broker Architecture (CORBA) minor code. The codes are listed in the article titled Troubleshooting the security components reference.

   For example, the following exception indicates a CORBA minor code of 49424300. The explanation of this error in the CORBA minor code table reads:

   `authentication failed error`

   In this case the user ID or password supplied by the client program is probably not valid:

```
org.omg.CORBA.NO_PERMISSION: Caught WSSecurityContextException in
WSSecurityContext.acceptSecContext(), reason: Major Code[0] Minor Code[0]
Message[ Exception caught invoking authenticateBasicAuthData from SecurityServer
for user jdoe. Reason: com.ibm.WebSphereSecurity.AuthenticationFailedException]
minor code: 49424300 completed:
No at com.ibm.ISecurityLocalObjectBaseL13Impl.PrincipalAuthFailReason.map_auth_fail_to_minor_code
(PrincipalAuthFailReason.java:83)
```

A CORBA INITIALIZE exception with CWWSA1477W: SECURITY CLIENT/SERVER CONFIGURATION MISMATCH error embedded, is received by client program from the server.

This error indicates that the security configuration for the server differs from the client in some fundamental way. The full exception message lists the specific mismatches. For example, the following exception lists three errors:

```
Exception received: org.omg.CORBA.INITIALIZE:
CWWSA1477W: SECURITY CLIENT/SERVER CONFIG MISMATCH:
The client security configuration (sas.client.props or outbound settings in
administrative console) does not support the server security configuration for
the following reasons:
ERROR 1: CWWSA0607E: The client requires SSL Confidentiality but the server does not
         support it.
ERROR 2: CWWSA0610E: The server requires SSL Integrity but the client does not
         support it.
ERROR 3: CWWSA0612E: The client requires client (e.g., userid/password or token),
         but the server does not support it.
     minor code: 0
     completed: No at
com.ibm.ISecurityLocalObjectBaseL13Impl.SecurityConnectionInterceptor.getConnectionKey
(SecurityConnectionInterceptor.java:1770)
```

In general, resolving the problem requires a change to the security configuration of either the client or the server. To determine which configuration setting is involved, look at the text following the CWWSA error message. For more detailed explanations and instructions, look in the message reference, by selecting the **Reference** view of the information center navigation and expanding **Messages** in the navigation tree.

In these particular cases:

- In ERROR 1, the client is requiring SSL confidentiality but the server does not support SSL confidentiality. Resolve this mismatch in one of two ways. Either update the server to support SSL confidentiality or update the client so that it no longer requires it.
- In ERROR 2, the server requires SSL integrity but the client does not support SSL integrity. Resolve this mismatch in one of two ways. Either update the server to support SSL integrity or update the client so that it no longer requires it.
- In ERROR 3, the client requires client authentication through a user id and password, but the server does not support this type of client authentication. Either the client or the server needs to change the configuration. To change the client configuration, modify the SAS.CLIENT.PROPS file for a pure client or change the outbound configuration for the server in the Security administrative console. To change the configuration for the target server, modify the inbound configuration in the Security administrative console.

Similarly, an exception like org.omg.CORBA.INITIALIZE: JSAS0477W: SECURITY CLIENT/SERVER CONFIG MISMATCH: appearing on the server trying to service a client request indicates a security configuration mismatch between client and server. The steps for resolving the problem are the same as for the JSAS1477W exceptions previously described.

Client program never gets prompted when accessing secured enterprise bean

Even though it seems that security is enabled and an enterprise bean is secured, occasions can occur when the client runs the remote method without prompting. If the remote method is protected, an authorization failure results. Otherwise, run the method as an unauthenticated user.

Possible reasons for this problem include:
- The server with which you are communicating might not have security enabled. Check with the WebSphere Application Server administrator to ensure that the server security is enabled. Access the security settings from within the **Security** section of the administrative console.
- The client does not have security enabled in the `sas.client.props` file. Edit the `sas.client.props` file to ensure the property com.ibm.CORBA.securityEnabled is set to `true`.
- The client does not have a ConfigURL specified. Verify that the property com.ibm.CORBA.ConfigURL is specified on the command line of the Java client, using the -D parameter.
- The specified ConfigURL does not have a valid URL syntax, or the `sas.client.props` that is pointed to cannot be found. Verify that the com.ibm.CORBA.ConfigURL property is valid. Check the Java documentation for a description of URL formatting rules. Also, validate that the file exists at the specified path.
- The client configuration does not support message layer client authentication (user ID and password). Verify that the `sas.client.props` file has one of the following properties set to `true`:
  - `com.ibm.CSI.performClientAuthenticationSupported=true`
  - `com.ibm.CSI.performClientAuthenticationRequired=true`
- The server configuration does not support message layer client authentication, which consists of a user ID and password. Check with the WebSphere Application Server administrator to verify that user ID and password authentication is specified for the inbound configuration of the server within the System Administration section of the administrative console administration tool.

Cannot stop an application server, node manager, or node after enabling security

If you use command-line utilities to stop WebSphere Application Server processes, apply additional parameters after enabling security to provide authentication and authorization information.

Use the `./stopServer -help` command to display the parameters to use.

Use the following command options after enabling security:
- `./stopServer serverName -username` *name* `-password` *password*
- `./stopNode -username` *name* `-password` *password*
- `./stopManager -username` *name* `-password` *password*

After enabling single sign-on, I cannot logon to the administrative console

This problem occurs when single sign-on (SSO) is enabled, and you attempt to access the administrative console using the short name of the server, for example `http://myserver:`*port_number*`/ibm/console`. The server accepts your user ID and password, but returns you to the logon page instead of the administrative console.

To correct this problem, use the fully qualified host name of the server, for example `http://myserver.mynetwork.mycompany.com:9060/ibm/console`.

The following exception displays in the SystemOut.log file after I start the server and enable security: ″SECJ0306E: No received or invocation credential exists on the thread.″

The following message displays when one or more nodes within the cell was not synchronized during configuration:

```
SECJ0306E: No received or invocation credential exists on the thread. The Role based
authorization check will not have an accessId of the caller to check. The parameters
are: access check method getServerConfig on resource FileTransferServer and module
FileTransferServer. The stack trace is java.lang.Exception: Invocation and received
credentials are both null.
```

Make sure that each of the nodes are synchronized and then restart the deployment manager.

Errors after configuring or enabling Secure Sockets Layer

This topic explains various problems you might encounter after configuring or enabling Secure Sockets Layer (SSL).

javax.net.ssl.SSLHandshakeException - The client and server could not negotiate the desired level of security. Reason: handshake failure

If you see a Java exception stack similar to the following example:

```
[Root exception is org.omg.CORBA.TRANSIENT:  CAUGHT_EXCEPTION_WHILE_CONFIGURING_
SSL_CLIENT_SOCKET: CWWJE0080E:   javax.net.ssl.SSLHandshakeException - The client
and server could not negotiate the desired level of   security. Reason: handshake
failure:host=MYSERVER,port=1079 minor code: 4942F303 completed: No]   at
com.ibm.CORBA.transport.TransportConnectionBase.connect
(TransportConnectionBase.java:NNN)
```

Some possible causes are:

- Not having common ciphers between the client and server.
- Not specifying the correct protocol.

To correct these problems:

1. Review the SSL settings. In the administrative console, click **Security > SSL certificate and key management**. Under Configuration settings, click **Manage endpoint security configurations >** *endpoint_configuration_name*. Under Related items, click **SSL configurations >** *SSL_configuration_name*. You can also browse the file manually by viewing the *app_server_root*/`properties/sas.client.props` file.
2. Check the property that is specified by the `com.ibm.ssl.protocol` file to determine which protocol is specified.
3. Check the cipher types that are specified by the com.ibm.ssl.enabledCipherSuites interface. You might want to add more cipher types to the list. To see which cipher suites are currently enabled, click **Quality of protection settings (QoP)**, and look for the **Cipher Suites** property.
4. Correct the protocol or cipher problem by using a different client or server protocol and cipher selection. Typical protocols are SSL or SSLv3.
5. Make the cipher selection 40-bit instead of 128-bit. For Common Secure Interoperability Version 2 (CSIv2), set both of the following properties to false in the sas.client.props file, or set security level=medium in the administrative console settings:
   - com.ibm.CSI.performMessageConfidentialityRequired=false
   - com.ibm.CSI.performMessageConfidentialitySupported=false

javax.net.ssl.SSLHandshakeException: unknown certificate

If you see a Java exception stack similar to the following example, it might be caused by not having the personal certificate for the server in the client truststore file:

```
ERROR: Could not get the initial context or unable to look up the starting context.
Exiting.  Exception received: javax.naming.ServiceUnavailableException: A
communication failure occurred while attempting to obtain an initial context using
the provider url: "corbaloc:iiop:localhost:2809". Make sure that the host and port
information is correct and that the server identified by the provider url is a
running name server. If no port number is specified, the default port number 2809
is used. Other possible causes include the network environment or workstation
network configuration. [Root exception is org.omg.CORBA.TRANSIENT:
CAUGHT_EXCEPTION_WHILE_CONFIGURING_SSL_CLIENT_SOCKET: CWWJE0080E:
javax.net.ssl.SSLHandshakeException - The client and server could not
negotiate the desired level of security. Reason: unknown
certificate:host=MYSERVER,port=1940 minor code: 4942F303 completed: No]
```

To correct this problem:

1. Check the client truststore file to determine if the signer certificate from the server personal certificate is there. For a self-signed server personal certificate, the signer certificate is the public key of the

personal certificate. For a certificate authority (CA)-signed server personal certificate, the signer certificate is the root CA certificate of the CA that signed the personal certificate.
2. Add the server signer certificate to the client truststore file.

javax.net.ssl.SSLHandshakeException: bad certificate

A Java exception stack error might display if the following situations occur:

- A personal certificate exists in the client keystore that is used for SSL mutual authentication.
- The signer certificate is not extracted into the server truststore file, and thus the server cannot trust the certificate whenever the SSL handshake is made.

The following message is an example of the Java exception stack error:

```
ERROR: Could not get the initial context or unable to look
up the starting context. Exiting.
Exception received: javax.naming.ServiceUnavailableException:
A communication failure occurred while attempting to obtain an
initial context using the provider url: "corbaloc:iiop:localhost:2809".
Make sure that the host and port information is correct and that the
server identified by the provider url is a running name
server. If no port number is specified, the default port number 2809
is used. Other possible causes include the network environment or
workstation network configuration.
[Root exception is org.omg.CORBA.TRANSIENT: CAUGHT_EXCEPTION_WHILE_CONFIGURING_SSL_
CLIENT_SOCKET: CWWJE0080E: javax.net.ssl.SSLHandshakeException - The client and
server could not negotiate the desired level of security. Reason:
bad certificate: host=MYSERVER,port=1940 minor code: 4942F303 completed: No]
```

To verify this problem, check the server truststore file to determine if the signer certificate from the client personal certificate is there. For a self-signed client personal certificate, the signer certificate is the public key of the personal certificate. For a certificate authority-signed client personal certificate, the signer certificate is the root CA certificate of the CA that signed the personal certificate.

To correct this problem, add the client signer certificate to the server truststore file.

org.omg.CORBA.INTERNAL: EntryNotFoundException or NTRegistryImp E CWSCJ0070E: No privilege id configured for: error when programmatically creating a credential

If you encounter the following exception in a client application attempting to request a credential from a WebSphere Application Server using SSL mutual authentication:

```
ERROR: Could not get the initial context or unable to look up the starting context.
Exiting. Exception received: org.omg.CORBA.INTERNAL: Trace from server: 1198777258
at host MYHOST on port 0 >>org.omg.CORBA.INTERNAL: EntryNotFoundException minor
code: 494210B0 completed:
No at com.ibm.ISecurityLocalObjectBaseL13Impl.PrincipalAuthFailReason.
map_auth_fail_to_minor_code(PrincipalAuthFailReason.java:99)
```

or a simultaneous error from the WebSphere Application Server that resembles:

```
[7/31/02 15:38:48:452 CDT] 27318f5 NTRegistryImp E CWSCJ0070E: No privilege id
configured for: testuser
```

The cause might be that the user ID sent by the client to the server is not in the user registry for that server.

To confirm this problem, check that an entry exists for the personal certificate that is sent to the server. Depending on the user registry mechanism, look at the native operating system user ID or Lightweight Directory Access Protocol (LDAP) server entries.

To correct this problem, add the user ID to the user registry entry (for example, operating system, LDAP directory, or other custom registry) for the personal certificate identity.

Single sign-on configuration troubleshooting tips

This topic describes common problems in configuring single sign-on (SSO) between a WebSphere Application Server and a Domino server and suggests possible solutions.

- Failure to save the Domino Web SSO configuration document

  The client must find Domino server documents for the participating SSO Domino servers. The Web SSO configuration document is encrypted for the servers that you specify. The home server that is indicated by the client location record must point to a server in the Domino domain where the participating servers reside. This pointer ensures that lookups can find the public keys of the servers.

  If you receive a message stating that one or more of the participating Domino servers cannot be found, then those servers cannot decrypt the Web SSO configuration document or perform SSO.

  When the Web SSO configuration document is saved, the status bar indicates how many public keys are used to encrypt the document by finding the listed servers, authors, and administrators in the document.

- Failure of the Domino server console to load the Web SSO configuration document at Domino HTTP server startup

  During configuration of SSO, the server document is configured for Multi-Server in the **Session Authentication** field. The Domino HTTP server tries to find and load a Web SSO configuration document during startup. The Domino server console reports the following information if a valid document is found and decrypted: `HTTP: Successfully loaded Web SSO Configuration`.

  If a server cannot load the Web SSO configuration document, SSO does not work. In this case, a server reports the following message: `HTTP: Error Loading Web SSO configuration`. Reverting to single-server session authentication.

  Verify that only one Web SSO configuration document is in the Web configurations view of the Domino directory and in the $WebSSOConfigs hidden view. You cannot create more than one document, but you can insert additional documents during replication.

  If you can verify only one Web SSO configuration document, consider another condition. When the public key of the server document does not match the public key in the ID file, this same error message can display. In this case, attempts to decrypt the Web SSO configuration document fail and the error message is generated.

  This situation can occur when the ID file is created multiple times, but the Server document is not updated correctly. Usually, an error message is displayed on the Domino server console stating that the public key does not match the server ID. If this situation occurs, SSO does not work because the document is encrypted with a public key for which the server does not possess the corresponding private key.

  To correct a key-mismatch problem:
  1. Copy the public key from the server ID file and paste it into the Server document.
  2. Create the Web SSO configuration document again.

- Authentication fails when accessing a protected resource.

  If a Web user is repeatedly prompted for a user ID and password, SSO is not working because either the Domino or the WebSphere Application Server security server cannot authenticate the user with the Lightweight Directory Access Protocol (LDAP) server. Check the following possibilities:
  - Verify that the LDAP server is accessible from the Domino server machine. Use the TCP/IP ping utility to check TCP/IP connectivity and to verify that the host machine is running.
  - Verify that the LDAP user is defined in the LDAP directory. Use the **idsldapsearch** utility to confirm that the user ID exists and that the password is correct. For example, you can run the following command, entered as a single line:

    You can use the OS/400 Qshell, a UNIX shell, or a Windows DOS prompt

    ```
    % ldapsearch -D "cn=John Doe, ou=Rochester, o=IBM, c=US" -w mypassword
    -h myhost.mycompany.com -p 389 -b "ou=Rochester, o=IBM, c=US" (objectclass=*)
    ```

The percent character (**%**) indicates the prompt and is not part of the command. A list of directory entries is expected. Possible error conditions and causes are contained in the following list:
  - No such object: This error indicates that the directory entry referenced by either the user's distinguished name (DN) value, which is specified after the **-D** option, or the base DN value, which is specified after the **-b** option, does not exist.
  - Credentials that are not valid: This error indicates that the password is not valid.
  - Cannot contact the LDAP server: This error indicates that the host name or the port specified for the server is not valid or that the LDAP server is not running.
  - An empty list means that the base directory that is specified by the **-b** option does not contain any directory entries.

– If you are using the user's short name or user ID instead of the distinguished name, verify that the directory entry is configured with the short name. For a Domino directory, verify the **Short name/UserID** field of the Person document. For other LDAP directories, verify the userid property of the directory entry.
– If Domino authentication fails when using an LDAP directory other than a Domino directory, verify the configuration settings of the LDAP server in the Directory assistance document in the Directory assistance database. Also verify that the Server document refers to the correct Directory assistance document. The following LDAP values that are specified in the Directory Assistance document must match the values specified for the user registry in the WebSphere Application Server administrative domain:
  - Domain name
  - LDAP host name
  - LDAP port
  - Base DN

  Additionally, the rules that are defined in the Directory assistance document must refer to the base distinguished name (DN) of the directory that contains the directory entries of the users.

  You can trace Domino server requests to the LDAP server by adding the following line to the server `notes.ini` file:

  ```
  webauth_verbose_trace=1
  ```

  After restarting the Domino server, trace messages are displayed in the Domino server console as Web users attempt to authenticate to the Domino server.

- Authorization failure when accessing a protected resource.

  After authenticating successfully, if an authorization error message is displayed, security is not configured correctly. Check the following possibilities:
  – For Domino databases, verify that the user is defined in the access-control settings for the database. Refer to the Domino administrative documentation for the correct way to specify the user's DN. For example, for the `DN cn=John Doe, ou=Rochester, o=IBM, c=US`, the value on the access-control list must be set as `John Doe/Rochester/IBM/US`.
  – For resources that are protected by WebSphere Application Server, verify that the security permissions are set correctly.
    - If granting permissions to selected groups, make sure that the user attempting to access the resource is a member of the group. For example, you can verify the members of the groups by using the following Web site to display the directory contents: `Ldap://myhost.mycompany.com:389/ou=Rochester, o=IBM, c=US??sub`
    - If you changed the LDAP configuration information (host, port, and base DN) in a WebSphere Application Server administrative domain since the permissions were set, the existing permissions are probably not valid and need to be recreated.

- SSO failure when accessing protected resources.

  If a Web user is prompted to authenticate with each resource, SSO is not configured correctly. Check the following possibilities:
  1. Configure both WebSphere Application Server and the Domino server to use the same LDAP directory. The HTTP cookie that is used for SSO stores the full DN of the user, for example, `cn=John Doe, ou=Rochester, o=IBM, c=US`, and the domain name service (DNS) domain.

2. Define Web users by hierarchical names if the Domino directory is used. For example, update the **User name** field in the Person document to include names of this format as the first value: `John Doe/Rochester/IBM/US`.
3. Specify the full DNS server name, not just the host name or TCP/IP address for Web sites issued to Domino servers and WebSphere Application Servers that are configured for SSO. For browsers to send cookies to a group of servers, the DNS domain must be included in the cookie, and the DNS domain in the cookie must match the Web address. This requirement is why you cannot use cookies across TCP/IP domains.
4. Configure both Domino and the WebSphere Application Server to use the same DNS domain. Verify that the DNS domain value is exactly the same, including capitalization. You need the name of the DNS domain in which WebSphere Application Server is configured. For additional information about configuring DNS domains for SSO, refer to the Single sign-on topic in the *Securing applications and their environment* PDF book.
5. Verify that the clustered Domino servers have the host name populated with the full DNS server name in the server document. By using the full DNS server name, Domino Internet Cluster Manager (ICM) can redirect to cluster members using SSO. If this field is not populated, by default, ICM redirects Web addresses to clustered Web servers by using the host name of the server only. ICM cannot send the SSO cookie because the DNS domain is not included in the Web address. To correct the problem:
   a. Edit the Server document.
   b. Click **Internet Protocols > HTTP** tab.
   c. Enter the full DNS name of the server in the **Host names** field.
6. If a port value for an LDAP server is specified for a WebSphere Application Server administrative domain, edit the Domino Web SSO configuration document and insert a backslash character (****) into the value of the **LDAP Realm** field before the colon character (**:**). For example, replace `myhost.mycompany.com:389` with `myhost.mycompany.com\:389`.

Enterprise Identity Mapping troubleshooting tips

The following information provides troubleshooting information for Enterprise Identity Mapping (EIM) configuration or connection factory configuration.

AdminControl service is not available

Symptom

The following message is displayed:

Message: WASX7017E: Exception received while running file ″/QIBM/ProdData/OS400/Java400/cfgIdToken.jacl″; exception information: com.ibm.ws.scripting.ScriptingException: AdminControl service not available.

Explanation

The application server or deployment manager of the WebSphere Application Server profile is not started, or the wsadmin option `-conntype NONE` is specified.

Configuration-related messages returned by the sample application to your Web browser session

Symptom

The following message is displayed:

Message: com.ibm.as400.access.AS400SecurityException: User ID is not known.

Explanation

The EIM does not contain a mapping for the user ID that is used to log in to the sample application.

Symptom

The following message is displayed:

Message: com.ibm.as400.access.ServerStartupException: Password encryption indicator is not valid.

Explanation

The target iSeries server is not configured for Enterprise Identity Mapping (EIM).

Symptom

The following message is displayed:

Message: java.net.ConnectException: A remote host refused an attempted connect operation.

Explanation

The target server is not an iSeries server.

Symptom

The following message is displayed:

Message: The lookup for the connection factory failed. Either the connector is not configured, or the servlet resource reference (JNDI name) is not set correctly in the web.xml file. The servlet expects the resource reference in the web.xml file to be eis/IdentityToken_Shared_Reference.

Explanation

Either the connector is not configured, or the servlet resource reference (JNDI name) is not set correctly in the `web.xml` file. The servlet expects the resource reference in the `web.xml` file to be `eis/IdentityToken_Shared_Reference`.

Symptom

The following message is displayed:

Message: The JAAS Subject object was not passed to the Java 2 Connector (J2C) connector because WebSphere Application Server security is not correctly configured for the servlet.

Explanation

WebSphere Application Server administrative security is not enabled.

Symptom

The following message is displayed:

Message: javax.resource.ResourceException: com.ibm.eim.jndi.DomainJNDI:method_name: failed to connect to initial directory context.

Explanation

This message is caused by one of the following issues:

- The authentication data entry that is configured for the connection factory contains an incorrect Lightweight Directory Access Protocol (LDAP) distinguished name.
- The authentication data entry that is configured for the connection factory contains an incorrect LDAP password.
- The LDAP host name that is configured for the connection factory is incorrect.
- The LDAP port that is configured for the connection factory is incorrect.
- The LDAP server is not started.
- The Enterprise Identity Mapping (EIM) domain name that is configured for the connection factory is incorrect.
- The EIM parent name that is configured for the connection factory is incorrect.

Symptom

The following message is displayed:

Message: javax.resource.ResourceException: Input URL is null or not valid.

Explanation

An LDAP host name is not configured for the connection factory.

Symptom	The following message is displayed: Message: com.ibm.as400.access.AS400SecurityException: An unknown problem occurred.
Explanation	The target iSeries server is not joined to the EIM domain that is configured for the connection factory, or the EIM source registry name is incorrect.

Perform the following steps to enable trace for EIM:

Note: This trace is only available for `idTokenRA.JCA15.rar`.

1. From the administrative console, select **Servers > Application Servers > server_name > Change Log Details Levels**.
2. Click the **Runtime** tab.
3. Select **Save runtime changes to configuration as well**.
4. Remove any previous entries in the text field, and type the following:
   `com.ibm.jca.idtoken.*=all: com.ibm.eim.token.*=all`
5. Click **Apply** and save the changes.

Authorization provider troubleshooting tips

This article describes the issues you might encounter using a Java Authorization Contract for Containers (JACC) authorization provider. Tivoli Access Manager is bundled with WebSphere Application Server as an authorization provider. However, you also can plug in your own authorization provider.

Tivoli Access Manager as a Java Authorization Contract for Containers authorization provider

You might encounter the following issues when using Tivoli Access Manager as a JACC authorization provider:

- The configuration of JACC might fail.
- The server might fail to start after configuring JACC.
- The application might not deploy properly.
- The startServer command might fail after you have configured Tivoli Access Manager or a clean uninstall did not take place after unconfiguring JACC.
- An ″HPDIA0202w An unknown user name was presented to Access Manager″ error might occur.
- An ″HPDAC0778E The specified user's account is set to invalid″ error might occur.
- An WASX7017E: Exception received while running file ″InsuranceServicesSingle.jacl″ error might occur.
- “Access denied exceptions accessing applications when using JACC” on page 355
- “An ″HPDBA0219E: An error occurred reading data from an SSL connection″ might occur” on page 356

External providers for Java Authorization Contract for Containers authorization provider

You might encounter the following issues when you use an external provider for JACC authorization:

- An ″HPDJA0506E Invalid argument: Null or zero-length user name field for the ACL entry″ error might occur.

The configuration of JACC might fail

If you have problems configuring JACC, check the following items:

- Ensure that the parameters are correct. For example, you do not want a number after TAM_Policy_server_hostname:7135, but you do want be a number after TAM_Authorization_server_hostname:7136 (for example, TAM_Authorization_server_hostname:7136:1).
- If a message such as “server can’t be contacted” is displayed, it is possible that the host names or port numbers of the Tivoli Access Manager servers are incorrect, or that the Tivoli Access Manager servers have not started.
- Ensure that the password for the sec_master user is correct.
- Check the `SystemOut.log` file and search for the `AMAS` string to see if any error messages are present.

The server might fail to start after configuring JACC

If the server does not start after JACC is configured, check the following items:

- Ensure that WebSphere Application Server and Tivoli Access Manager use the same Lightweight Directory Access Protocol (LDAP) server.
- If the message “Policy Director Authentication failed″ is displayed, ensure that the:
  - WebSphere Application Server LDAP server ID is the same as the “Administrator user” in the Tivoli Access Manager JACC configuration panel.
  - Verify that the Tivoli Access Manager Administrator distinguished name (DN) is correct.
  - Verify that the password of the Tivoli Access Manager administrator has not expired and is valid.
  - Ensure that the account is valid for the Tivoli Access Manager administrator.
- If a message such as `socket can't be opened for` *xxxx* (where *xxxx* is a number) is displayed, take the following actions:
  1. Go to the *profile_root*`/etc/tam` directory.
  2. Change *xxxx* to an available port number in the `amwas.commomconfig.properties` file, and the `amwas*cellName_dmgr.properties` file if the deployment manager failed to start. If the node failed to start, change *xxx* to an available port number in the `amwas*cellName_nodeName_.properties` file. If the Application Server failed to start, change *xxxx* in the `amwas*cellname_nodeName_serverName.properties` file.

The application might not deploy properly

When you click **Save**, the policy and role information is propagated to the Tivoli Access Manager policy. This process might take some time to finish. If the save fails, you must uninstall the application and then reinstall it.

To access an application after it is installed, you must wait 30 seconds, by default, to start the application after you save.

The startServer command might fail after you configure Tivoli Access Manager or a clean uninstall did not take place after unconfiguring JACC.

If the cleanup for JACC unconfiguration or start server fails after JACC is configured, take the following actions:

- Remove Tivoli Access Manager properties files from WebSphere Application Server. For each application server in a Network Deployment (ND) environment with N servers defined (for example, server1, server2).

  The following files must be removed.

  profile_root`/etc/pd/PolicyDirector/PDPerm.properties`
  profile_root`/etc/pd/PolicyDirector/PdPerm.ks`
  profile_root`/etc/tam/*`
- Use a utility to clear the security configuration and return the system to the state it was in before you configure the JACC provider for Tivoli Access Manager. The utility removes all of the

PDLoginModuleWrapper entries as well as the Tivoli Access Manager authorization table entry from the `security.xml` file, effectively removing the JACC provider for Tivoli Access Manager. Backup the `security.xml` file before running this utility.

Enter the following commands:

```
java -Djava.version=1.5 -classpath
"app_server_root/lib/AMJACCProvider.jar:CLASSPATH"
com.tivoli.pd.as.jacc.cfg.CleanSecXML fully_qualified_path/security.xml
```

An ″HPDIA0202w An unknown user name was presented to Access Manager″ error might occur

You might encounter the following error message if you try to use an existing user in a Local Directory Access Protocol (LDAP) user registry with Tivoli Access Manager:

```
AWXJR0008E   Failed to create a PDPrincipal for principal mgr1.:
AWXJR0007E   A Tivoli Access Manager exception was caught. Details are:
"HPDIA0202W  An unknown user name was presented to Access Manager."
```

This problem might be caused by the host name exceeding predefined limits with Tivoli Access Manager when it is configured against MS Active Directory. In WebSphere Application Server, the maximum length of the host name can not exceed 46 characters.

Check that the host name is not fully qualified. Configure the machine so that the host name does not include the host domain.

To correct this error, complete the following steps:

1. On the command line, type the following information to get a Tivoli Access Manager command prompt:

   ```
   pdadmin -a administrator_name -p administrator_password
   ```

   The pdadmin *administrator_name* prompt is displayed. For example:

   ```
   pdadmin -a administrator1 -p passw0rd
   ```

2. At the pdadmin command prompt, import the user from the LDAP user registry to Tivoli Access Manager by typing the following information:

   ```
   user import user_name cn=user_name,o=organization_name,c=country
   ```

   For example:

   ```
   user import jstar cn=jstar,o=ibm,c=us
   ```

After importing the user to Tivoli Access Manager, you must use the **user modify** command to set the user account to `valid`. The following syntax shows how to use this command:

```
user modify user_name account-valid yes
```

For example:

```
user modify jstar account-valid yes
```

For information on how to import a group from LDAP to Tivoli Access Manager, see the Tivoli Access Manager documentation.

An ″HPDAC0778E The specified user’s account is set to invalid″ error might occur

You might encounter the following error message after you import a user to Tivoli Access Manager and restart the client:

```
AWXJR0008E    Failed to create a PDPrincipal for principal mgr1.:
AWXJR0007E    A Tivoli Access Manager exception was caught.
Details are: "HPDAC0778E   The specified user's account is set to invalid."
```

To correct this error, use the `user modify` command to set the user account to valid. The following syntax shows how to use this command:

```
user modify user_name account-valid yes
```

For example:

```
user modify jstar account-valid yes
```

An ″HPDJA0506E Invalid argument: Null or zero-length user name field for the ACL entry″ error might occur

You might encounter an error similar to the following message when you propagate the security policy information from the application to the provider using the wsadmin **propagatePolicyToJACCProvider** command:

```
AWXJR0035E   An error occurred while  attempting to add member,
               cn=agent3,o=ibm,c=us, to role AgentRole
HPDJA0506E   Invalid argument: Null or zero-length user name field for
               the ACL entry
```

To correct this error, create or import the user, that is mapped to the security role to the Tivoli Access Manager. For more information on propagating the security policy information, see the documentation for your authorization provider.

An WASX7017E: Exception received while running file ″InsuranceServicesSingle.jacl″ error might occur

After the JACC provider and Tivoli Access Manager are enabled, when attempting to install the application, which is configured with security roles using the wsadmin command, the following error might occur:

```
WASX7017E: Exception received while running file "InsuranceServicesSingle.jacl";
exception information: com.ibm.ws.scripting.ScriptingException: WASX7111E:
Cannot find a match for supplied option:
"[RuleManager, , , cn=mgr3,o=ibm,c=us|cn=agent3,o=ibm,c=us, cn=ManagerGro
up,o=ibm,c=us|cn=AgentGroup,o=ibm,c=us]" for task "MapRolesToUsers
```

The $AdminApp MapRolesToUsers task option is no longer valid when Tivoli Access Manager is used as the authorization server. To correct the error, change MapRolesToUsers to TAMMapRolesToUsers.

Access denied exceptions accessing applications when using JACC

In the case of Tivoli Access Manager, you might see the following error message.

```
AWXJR0044E: The access decision for Permission, {0}, was denied because either the
PolicyConfiguration or RoleConfiguration objects did not get created successfully at
application installation time. RoleConfiguration exists = {false}, PolicyConfiguration
exists = {false}."
```

If the access denied exceptions are not expected for the application, check the SystemOut.log files to see if the security policy information was correctly propagated to the provider.

If the security policy information for the application is successfully propagated to the provider, the audit statements with the message key SECJ0415I appear. However, if there was a problem propagating the security policy information to the provider (for example: network problems, JACC provider is not available), the SystemOut.log files contain the error message with the message keys SECJ0396E (during install) or SECJ0398E (during modification). The installation of the application is not stopped due to a failure to propagate the security policy to the JACC provider. Also, in the case of failure, no exception or error messages appear during the save operation. When the problem causing this failure is fixed, run the propagatePolicyToJaccProvider tool to propagate the security policy information to the provider without reinstalling the application. For more information about this task, see the Propagating security policy of

installed applications to a JACC provider using wsadmin scripting topic in the *Securing applications and their environment* PDF book.

An ″HPDBA0219E: An error occurred reading data from an SSL connection″ might occur

An error message (HPDBA0219E) might appear in dmgr SystemOut.log when you install an application on WebSphere Application Server for Network Deployment (ND) and a managed node with Tivoli Access Manager is enabled.

If the error occurs, then the security policy data of recently deployed applications might not be immediately available. The policy data is available based on the server replicate time of the Tivoli Access Manager. This is defaulted to 30 seconds after all updates have been completed. To ensure that the latest policy data is available, log on to the pdadmin console and type: `server replicate`.

Password decoding troubleshooting tips

If the password encoding is corrupted and you cannot decode a password, you can complete one of the following tasks.

- If the password is contained in a server configuration file, edit the file and set the password to the clear text value. After changing the value, restart the server.
- If the password is contained in the `sas.client.props` file or the `soap.client.props` file, edit the file and set the password to the appropriate clear text value. After changing the value, use the PropFilePasswordEncoder utility to encode the password. For more information on the PropFilePasswordEncoder utility, see thePropFilePasswordEncoder command reference.

The profile-specific setupCmdLine QShell script contains a property that you can use to obtain trace information when using the OS400 algorithm with Java clients and administrative commands for WebSphere Application Server. To obtain the trace, set the os400.security.password.debug property to `true`. The trace is printed to standard output.

SPNEGO trust association interceptor (TAI) troubleshooting tips

Presented here is a list of trouble shooting tips useful in diagnosing Simple and Protected GSS-API Negotiation (SPNEGO) TAI problems and exceptions.

The IBM Java Generic Security Service (JGSS) and IBM SPNEGO providers use a Java virtual machine (JVM) custom property to control trace information. The SPNEGO TAI uses the JRas facility to allow an administrator to trace only specific classes. To debug the TAI using tracing, the following important trace specifications or JVM customer should be used:

Table 4. SPNEGO TAI trace specifications

Trace	Use
com.ibm.security.jgss.debug	Set this **JVM Custom Property** to `all` to trace through JGSS code. Messages appear in the `trace.log` file, and **SystemOut.log**.
com.ibm.security.krb5.Krb5Debug	Set this **JVM Custom Property** to `all` to trace through the Kerberos5-specific JGSS code. Messages appear in the `trace.log` file, and **SystemOut.log**.
*com.ibm.ws.security.spnego.**	Set this trace on using the **administrative console > troubleshooting > Logging and Tracing > server1 > Change Log Detail Levels > com.ibm.ws.security.spnego.***. Messages appear in the `trace.log` file.

Problem: WebSphere Application Server and the Active Directory (AD) Domain Controller's time are not synchronized within 5 minutes

The time is not synchronized between WebSphere Application Server and AD Domain Controller.

```
[2/24/06 13:12:46:093 CST] 00000060 Context       2 com.ibm.ws.security.spnego.Context
    begin GSSContext accepted
[2/24/06 13:12:46:093 CST] 00000060 Context       E com.ibm.ws.security.spnego.Context
    begin
CWSPN0011E: An invalid SPNEGO token has been encountered while authenticating a
    HttpServletRequest:
0000:  60820160 06062b06 01050502 a1820154    `..` ..+. .... ...T
0010:  30820150 a0030a01 01a10b06 092a8648    0..P .... .... .*.H
0020:  82f71201 0202a282 013a0482 01366082    .... .... .:.. .6`.
0030:  01320609 2a864886 f7120102 0203007e    .2.. *.H. .... ...~
0040:  82012130 82011da0 03020105 a1030201    ..!0 .... .... ....
0050:  1ea41118 0f323030 36303232 34313931    .... .200 6022 4191
0060:  3234365a a5050203 016b48a6 03020125    246Z .... .kH. ...%
0070:  a9161b14 57535345 432e4155 5354494e    .... WSSE C.AU STIN
0080:  2e49424d 2e434f4d aa2d302b a0030201    .IBM .COM .-0+ ....
0090:  00a12430 221b0448 5454501b 1a773230    ..$0 "..H TTP. .w20
00a0:  30337365 63646576 2e617573 74696e2e    03se cdev .aus tin.
00b0:  69626d2e 636f6dab 81aa1b81 a76f7267    ibm. com. .... .org
00c0:  2e696574 662e6a67 73732e47 53534578    .iet f.jg ss.G SSEx
00d0:  63657074 696f6e2c 206d616a 6f722063    cept ion,  maj or c
00e0:  6f64653a 2031302c 206d696e 6f722063    ode:  10,  min or c
00f0:  6f64653a 2033370a 096d616a 6f722073    ode:  37. .maj or s
0100:  7472696e 673a2044 65666563 74697665    trin g: D efec tive
0110:  20746f6b 656e0a09 6d696e6f 72207374     tok en.. mino r st
0120:  72696e67 3a20436c 69656e74 2074696d    ring : Cl ient  tim
0130:  65204672 69646179 2c204665 62727561    e Fr iday , Fe brua
0140:  72792032 342c2032 30303620 61742031    ry 2 4, 2 006  at 1
0150:  3a31323a 34352050 4d20746f 6f20736b    :12: 45 P M to o sk
0160:  65776564                               ewed
```

Solution: You can fix this in one of two ways. The preferred way is to synchronize the WebSphere Application Server system time to within 5 minutes of the AD server's time. A best practice is to use a time server to keep all systems synchronized. Or you can add or adjust the clockskew parameter in the Kerberos configuration file.

Note: The default for the clockskew parameter is 300 seconds (or 5 minutes).

Problem: Getting exception: No factory available to create a name for mechanism 1.3.6.1.5.5.2

There apparently is no factory available to process the creation of a name for the specific mechanism.

The `systemout.log` file displays something like this:

```
[4/8/05 22:51:24:542 EDT] 5003e481 SystemOut     O [JGSS_DBG_PROV] Provider
        IBMJGSSProvider version 1.01 does not support mech 1.3.6.1.5.5.2
[4/8/05 22:51:24:582 EDT] 5003e481 ServerCredent >
        com.ibm.ws.security.spnego.ServerCredential initialize ENTRY
SPNEGO014: Kerberos initialization Failure: org.ietf.jgss.GSSException, major code: 2,
        minor code: 0
 major string: Unsupported mechanism
 minor string: No factory available to create name for mechanism 1.3.6.1.5.5.2
 at com.ibm.security.jgss.i18n.I18NException.throwGSSException
        (I18NException.java:30)
 at com.ibm.security.jgss.GSSManagerImpl.a(GSSManagerImpl.java:36)
 at com.ibm.security.jgss.GSSCredentialImpl.add(GSSCredentialImpl.java:217)
 at com.ibm.security.jgss.GSSCredentialImpl.<init>(GSSCredentialImpl.java:264)
```

Solution: Check the `java.security` file to ensure it contains the IBMSPNEGO security provider and that the provider is defined correctly. The `java.security` file should contain a line similar to:

```
security.provider.6=com.ibm.security.jgss.mech.spnego.IBMSPNEGO
```

Problem: Getting an exception

An exception has occurred when reporting to the client.

You get the following display.

```
Error authenticating request. Reporting to client
Major code = 11, Minor code = 31
org.ietf.jgss.GSSException, major code: 11, minor code: 31
 major string: General failure, unspecified at GSSAPI level
 minor string: Kerberos error while decoding and verifying token:
                com.ibm.security.krb5.internal.KrbException, status code: 31
 message: Integrity check on decrypted field failed
```

as the JGSS library is trying to process the SPNEGO token.

Cause: This exception is the result of encoding the ticket using one key and attempting to decode it using a different key. There are number of possible reasons for this condition:

1. The Kerberos keytab file has not been copied to the server machine after it has been regenerated.
2. The Kerberos configuration points to the wrong Kerberos keytab file.
3. The Kerberos service principal name (SPN) has been defined to the Active Directory more than once; this can occur because you have another userid with a similarly defined SPN (either exactly the same name, or one having a different name but with a port defined part of the SPN).

Solution: If the problem is with the Kerberos keytab file, then fix it. If the problem is with multiple SPN definitions, then remove the extra or conflicting SPN, confirm that the SPN is no longer registered with the Active Directory, and then add the SPN. The Active Directory may need to be searched for other entries with SPNs defined that clash with the SPN.

To confirm that the SPN is not registered, the command:

```
setspn –l userid
```

should return with the following response:

```
Cannot find account userid
```

Problem: Single sign-on is not occurring.

When trace is turned on, the following message appears:

```
[2/27/06 14:28:04:191 CST] 00000059 SpnegoHandler <
         com.ibm.ws.security.spnego.SpnegoHandler handleRequest: Received a
         non-SPNEGO Authorization Header RETURN
```

Cause: The client is returning an NT LAN manager (NTLM) response to the authorize challenge, not a SPNEGO token. This condition can be occur due to any of the following reasons:

- The client has not been configured properly.
- The client is not using a supported browser. For example, when using Microsoft Internet Explorer 5.5, SP1 responds with a non-SPNEGO authentication header.
- The user has not logged into the Active Directory domain, or into a trusted domain, or the client used does not support integrated authentication with Windows – in this case, the SPNEGO TAI is working properly.
- The user is accessing a service defined on the same machine upon which the client is running (local host). Microsoft Internet Explorer resolves the host name of the URL to `http://localhostsomeURL` instead of a fully qualified name.

- The SPN is not found in the Active Directory. The SPN must be of the format `HTTP/server.realm.com`. The command to add the SPN is

  ```
  setspn –a HTTP/server.realm.com userid
  ```

If the SPN is defined incorrectly as `HTTP/server.realm.com@REALM.COM` with the addition of `@REALM.COM`, then delete the user, redefine the user, and redefine the SPN.

Problem: Credential Delegation is not working

An invalid option is detected. When trace is turned on, the following message is displayed:

```
com.ibm.security.krb5.KrbException, status code: 101 message: Invalid option in
ticket request
```

Cause: The Kerberos configuration file is not properly configured.

Solution: Ensure that neither renewable, nor proxiable are set to true.

Problem: Unable to get SSO working using RC4-HMAC encryption.

When trace is turned on, you get the following message in the trace:

```
com.ibm.security.krb5.internal.crypto.KrbCryptoException, status code: 0
 message: Checksum error; received checksum does not match computed checksum
```

Cause: RC4-HMAC encryption is not supported with a Microsoft Windows 2000 Kerberos key distribution center (KDC). To confirm this condition, examine the trace and identify where the exception is thrown. The content of the incoming ticket should be visible in the trace. Although the incoming ticket is encrypted, the SPN for the service is readable. If a Microsoft Windows 2000 KDC is used and the system is configured to use RC4-HMAC, the string representing the ticket for `userid@REALM` (instead of the expected `HTTP/hostname.realm@REALM`) is displayed. For example, this is beginning of the ticket received from a Microsoft Windows 2000 KDC:

```
0000: 01 00 6e 82 04 7f 30 82  04 7b a0 03 02 01 05 a1  ..n...0.........
0010: 03 02 01 0e a2 07 03 05  00 20 00 00 00 a3 82 03  ................
0020: a5 61 82 03 a1 30 82 03  9d a0 03 02 01 05 a1 0a  .a...0..........
0030: 1b 08 45 50 46 44 2e 4e  45 54 a2 18 30 16 a0 03  ...REALM.COM.0..
0040: 02 01 01 a1 0f 30 0d 1b  0b 65 70 66 64 77 61 73  .....0...userid
0050: 75 6e 69 74 a3 82 03 6e  30 82 03 6a a0 03 02 01  .a.f...n0..j....
```

The realm is REALM.COM. The service name is userid. A correctly formed ticket for the same SPN is:

```
0000: 01 00 6e 82 04 56 30 82  04 52 a0 03 02 01 05 a1  ..n..V0..R......
0010: 03 02 01 0e a2 07 03 05  00 20 00 00 00 a3 82 03  ................
0020: 82 61 82 03 7e 30 82 03  7a a0 03 02 01 05 a1 0a  .a...0..z.......
0030: 1b 08 45 50 46 44 2e 4e  45 54 a2 2a 30 28 a0 03  ..REALM.COM.0...
0040: 02 01 02 a1 21 30 1f 1b  04 48 54 54 50 1b 17 75  .....0...HTTP..u
0050: 73 31 30 6b 65 70 66 77  61 73 73 30 31 2e 65 70  serid.realm.com.
0060: 66 64 2e 6e 65 74 a3 82  03 39 30 82 03 35 a0 03  ...n.....90..5..
```

Solution: To correct the problem, either use the Single data encryption standard (DES) or use a Microsoft Windows 2003 Server for a KDC. Remember to regenerate the SPN, and the Kerberos keytab file.

Problem: User receives the following message when accessing a protected URL through the SPNEGO SSO

```
Bad Request

Your browser sent a request that this server could not understand.
Size of request header field exceeds server limit.

Authorization: Negotiate YII......
```

Cause: This message is generated by the Apache/IBM HTTP Server. This server is indicating that the authorization header returned by the user's browser is too large. The long string that follows the word Negotiate (in the error message above) is the SPNEGO token. This SPNEGO token is a wrapper of the Microsoft Windows Kerberos token. Microsoft Windows includes the user's PAC information in the Kerberos token. The more security groups that the user belongs to, the more PAC information is inserted in the Kerberos token, and the larger the SPNEGO becomes. IBM HTTP Server 2.0 (also Apache 2.0 and IBM HTTP Server 6.0) limit the size of any acceptable HTTP header to be 8K. In Microsoft Windows domains having many groups, and with user membership in many groups, the size of the user's SPNEGO token may exceed the 8K limit.

Solution: If possible, reduce the number of security groups the user is a member of. IBM HTTP Server 2.0.47 cumulative fix PK01070 allows for HTTP header sizes up to and beyond the Microsoft limit of 12K. WebSphere Application Server Version 6.0 users can obtain this fix in fixpack 6.0.0.2.

Note: Non-Apache based Web servers may require differing solutions.

Problem: Even with JGSS tracing disabled, some KRB_DBG_KDC messages appear in the SystemOut.log

Cause: While most of the JGSS tracing is controlled by the com.ibm.security.jgss.debug property, a small set of messages are controlled by the com.ibm.security.krb5.Krb5Debug property. The com.ibm.security.krb5.Krb5Debug property has a default value to put some messages to the **SystemOut.log**.

Solution: To remove all KRB_DBG_KDC messages from the **SystemOut.log**, set the JVM property as follows:

```
-Dcom.ibm.security.krb5.Krb5Debug=none
```

.

Problem: HTTP Post parameters are lost during interaction with the SPNEGO TAI, when stepping down to userid/password login.

Cause: The Microsoft Internet Explorer maintains state during a user's request. If a request was given the response of an ″HTTP 401 Authenticate Negotiate″, and the browser responds with a NTLM token obtained through a userid/password challenge, the browser resubmits the request. If this second request is given a response of an HTML page containing a redirection to the same URL but with new arguments (via Javascript) then the browser does not resubmit the POST parameters. To avoid this problem, it is critical to NOT perform the automatic redirection. If the user clicks on a link, the problem does not occur. See section 5.2 Client Returns NTLM Token to SPNEGO Challenge for a resolution to the problem,

Solution: The browser responds to the Authenticate/Negotiate challenge with an NTLM token, not an SPNEGO token. The SPNEGO TAI sees the NTLM, and returns back a HTTP 403 response, along with the HTML page. When the browser runs the Javascript redirTimer function, any POST of GET parameters that were present on the original request are lost.

By leveraging the SPN<id>.NTLMTokenReceivedPage property, an appropriate message page can be returned to the user. The default message that is returned (in the absence of a user defined property) is:

```
"<html><head><title>An NTLM Token was Received.</title></head>"
 + "<body>Your browser configuration is correct, but you have not logged into
          a supported Windows Domain."
 + "<p>Please login to the application using the normal login page.</html>";
```

Using the SPN<id>.NTLMTokenReceivedPage property, you can customize the exact response. It is critical that the returned HTML not perform a redirection.

When the SPNEGO TAI has been configured to use the shipped default HTTPHeaderFilter class as the SPN<id>.filterClass, then the SPN<id>.filter can be used to allow the second request to flow directly to the normal WebSphere Application Server security mechanism. In this way, the user experiences the normal authentication mechanism.

An example of such a configuration follows. The required SPNEGO TAI properties necessary and the HTML file content are presented.

Table 5. SPNEGO TAI properties and HTML

SPNEGO TAI Property Name	HTML File Content
com.ibm.ws.security.spnego.SPN1.hostName	server.wasteched30.torolab.ibm.com
com.ibm.ws.security.spnego.SPN1.filterClass	com.ibm.ws.security.spnego.HTTPHeaderFilter
com.ibm.ws.security.spnego.SPN1.filter	request-url!=noSPNEGO
com.ibm.ws.security.spnego.SPN1.NTLMTokenReceivedPage	File:///C:/temp/NTLM.html

Note: Observe that the filter property instructs the SPNEGO TAI to NOT intercept any HTTP request that contains the string “noSPNEGO”.

Here is an example of a generating a helpful response.

```
<html>
<head>
<title>NTLM Authentication Received </title>
<script language="javascript">
 var purl=""+document.location;
if (purl.indexOf("noSPNEGO")<0) {
  if(purl.indexOf('?')>=0) purl+="&noSPNEGO";
  else purl+="?noSPNEGO";
 }
</script>
</head>
<body>
<p>An NTLM token was retrieved in response to the SPNEGO challenge. It is likely that
you are not logged into a Windows domain.<br>
Click on the following link to get the requested website.
<script language="javascript">
 document.write("<a href='"+purl+"'>");
 document.write("Open the same page using the normal authentication
  mechanism.");
 document.write("</a><br>");
</script>
You will not automatically be redirected.
</body>
</html>
```

Chapter 24. Naming and directory

Troubleshooting name space problems

When developing or running applications, you might encounter name space problems.

Many naming problems can be avoided by fully understanding the key underlying concepts of WebSphere Application Server naming.

1. Review the key concepts of WebSphere Application Server naming, especially the sections about Name space logical view and Lookup names support in deployment descriptors and thin clients in the *Developing and deploying applications* PDF book.
2. Review the programming examples that are included in the sections explaining the JNDI and CosNaming interfaces in the *Developing and deploying applications* PDF book.
3. Read "Naming service troubleshooting tips" for additional general information.
4. Read "Cannot look up an object hosted by WebSphere Application Server from a servlet, JSP file, or other client" on page 364 for information on lookup errors.

Naming service troubleshooting tips

Naming is a J2EE service which publishes and provides access to resources such as connection pools, enterprise beans, and message listeners to client processes. If you have problems in accessing a resource which otherwise appears to be healthy, the naming service might be involved.

To investigate problems with the WebSphere Application Server Naming service:

- Browse the JVM logs for the server which is hosting the resource you are trying to access. Messages starting with `NMSV` are related to the Naming Service.
- With WebSphere Application Server running, run the **dumpNameSpace** Qshell script for OS/400 systems, the **dumpNameSpace** command for Windows systems, or the **dumpNameSpace.sh** command for operating systems such as AIX or Linux, and pipe, redirect, or ″more″ the output so that it is easily viewed. This command results in a display of objects in the WebSphere Application Server namespace, including the directory path and object name.

  Remember: The **dumpNameSpace** command does not dump all of the objects in the distributed namespace. It only dumps the objects that are in the local namespace of the process against which the command was run.
- If the object a client needs to access does not appear, use the administrative console to verify that:
  - The server hosting the target resource is started.
  - The Web module or EJB container, if applicable, hosting the target resource is running.
  - The JNDI name of the target resource is correct and updated.
  - If the problem resource is remote, that is, not on the same node as the Name Server node, that the JNDI name is fully qualified, including the host name. This is especially applicable to Network Deployment configurations
- View detailed information on the runtime behavior of the WebSphere Application Server Naming service by enabling trace on the following components and reviewing the output:
  - com.ibm.ws.naming.*
  - com.ibm.websphere.naming.*
- If you see an exception that appears to be CORBA related (″CORBA″ appears as part of the exception name) look for a naming-services-specific CORBA minor code, further down in the exception stack, for information on the real cause of the problem. For a list of naming service exceptions and explanations, see the class com.ibm.websphere.naming.WsnCorbaMinorCodes in the API documentation that is included in the Reference section of the information center.

If none of these steps solve the problem:

- For specific problems that can cause access to named object hosted in WebSphere Application Server to fail, see Cannot look up an object hosted by WebSphere Application Server from a servlet, JSP file, or other client.
- Check to see if the problem has been identified and documented using the links in Diagnosing and fixing problems: Resources for learning.

For current information available from IBM Support on known problems and their resolution, see the IBM Support page.

IBM Support has documents that can save you time gathering information needed to resolve this problem. Before opening a PMR, see the IBM Support page.

Cannot look up an object hosted by WebSphere Application Server from a servlet, JSP file, or other client

To resolve problems encountered when a servlet, JavaServer Pages file, standalone application or other client attempts to access an enterprise bean, ConnectionPool, or other named object hosted by WebSphere Application Server, you must first verify that the target server can be accessed from the client.

Follow these steps:

- From a command prompt on the client's server, enter *"ping server_name"* and verify connectivity.
- Use the WebSphere Application Server administrative console to verify that the target resource's application server and, if applicable, EJB module or Web module, is started.

Continue only if there is no problem with connectivity and the target resource appears to be running.

What kind of error are you seeing?

- "NameNotFoundException from JNDI lookup operation "
- "CannotInstantiateObjectException from JNDI lookup operation " on page 365
- "Message NMSV0610I appears in the server's log file, indicating that some Naming exception has occurred " on page 365
- "OperationNotSupportedException from JNDI Context operation" on page 366
- "WSVR0046E: Failed to bind, ejb/jndiName: ejb/jndiName. Original exception : org.omg.CosNaming.NamingContextPackage.AlreadyBound " on page 366
- "ConfigurationException from "new InitialContext" operation or from a JNDI Context operation with a URL name" on page 366
- "ServiceUnavailableException from "new InitialContext" operation" on page 367
- "CommunicationException thrown from a "new InitialContext" operation" on page 367
- NMSV0605E: A Reference object looked up from the context. See the *Developing and deploying applications* PDF for more information.

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

NameNotFoundException from JNDI lookup operation

There are three causes for a NameNotFoundException:

- The lookup name is incorrect.
- The object being looked up is not bound.
- Two servers with the same name running on the same host are being used to interoperate.

Incorrect lookup name

If you encounter a NameNotFoundException when trying to access an enterprise bean, data source, messaging resource, or other resource:

1. Determine the cause of the NameNotFoundException.

Browse the properties of the target object in the administrative console, and verify that the JNDI name it specifies matches the JNDI name the client is using.

If you are looking up an object that resides on a server different from the one from which the initial context was obtained, you must use the fully qualified name.

- If access is from another server object such as a servlet accessing an enterprise bean and you are using the default context, not specifying the fully qualified JNDI name, you might get a NameNotFoundException if the object is hosted on a different server.
- If access is from a standalone client, it might be that the object you are attempting access is on a server different from the server from which you obtained the initial context.

2. Use the fully-qualified JNDI name to correct the problem.

Object being looked up is not bound

To correct a NameNotFoundException where the object being looked up is not bound:

1. If the object being looked up is an application object such as an enterprise bean, ensure that the application is running.
2. Run the dumpNameSpace tool to view the contents of the name space to verify that the object being looked up is bound to the name space with the expected name.

Two servers with the same name running on the same host are being used to interoperate

For a server application to access objects on a server in a different node running on the same host, the names of the two servers must be different. For example, if an application running on a server in node1 looks up an object that resides on a server in node2, and node1 and node2 are installed on the same host, the server names must be different. If both server names are the same (for example, server1), the lookup fails with a NameNotFoundException.

To correct the problem, use different server names, such as server1 and server2.

Note that any remote object references from server1 in node1 to server1 in node2 in the described configuration fail. JNDI lookups fail with a NameNotFoundException, but object references obtained through means other than JNDI lookups also fail, most likely with an org.omg.CORBA.OBJECT_NOT_EXIST exception. Object references between servers in different nodes and on different hosts do not result in an exception even if the server names match.

CannotInstantiateObjectException from JNDI lookup operation

If you encounter this exception in trying to access an enterprise bean, data source, messaging resource, or other resource, possible causes include:

- A serialized Java object is being looked up, but the necessary classes required to deserialize it are not in the runtime environment.
- A Reference object is being looked up, and the associated factory used to process it as part of the lookup processing is failing.

To determine the precise cause of the problem:

- Look at the relevant logs for exceptions immediately preceding the CannotInstantiateObjectException. If it is a java.lang.NoClassDefFoundError or java.lang.ClassNotFoundException, make sure the class referenced in the error message can be located by the class loader. See the section Class loading in the *Developing and deploying applications* PDF book.

  View the JVM logs.
- Print out the stack trace for the root cause and look for the factory class. It will be called by `javax.naming.NamingManager.getObjectInstance()`. The reason for the failure will depend on the factory implementation, and may require you to contact the developer of the factory class.

Message NMSV0610I appears in the server's log file, indicating that some Naming exception has occurred

This error is informational only and is provided in case the exception is related to an actual problem. Most of the time, it is not. If it is, the log file should contain adjacent entries to provide context.

- If no problems are being experienced, ignore this message. Also ignore the message if the problem you are experiencing does not appear to be related to the exception being reported and if there are no other adjacent error messages in the log.
- If a problem is being experienced, look in the log for underlying error messages.
- The information provided in message NMSV0610I can provide valuable debug data for other adjacent error messages posted in response to the Naming exception that occurred.

OperationNotSupportedException from JNDI Context operation

This error has two possible causes:
- An update operation, such as a bind, is being performed with a name that starts with "java:comp/env". This context and its subcontexts are read-only contexts.
- A Context bind or rebind operation of a non-CORBA object is being performed on a remote name space that does not belong to WebSphere Application Server. Only CORBA objects can be bound to these CosNaming name spaces.

To determine which of these errors is causing the problem, check the full exception message.

WSVR0046E: Failed to bind, ejb/jndiName: ejb/jndiName. Original exception : org.omg.CosNaming.NamingContextPackage.AlreadyBound

This error occurs two enterprise bean server applications were installed on the same server such that a binding name conflict occurred. That is, a jndiName value is the same in the two applications' deployment descriptors. The error will surface during server startup when the second application using that jndiName value is started.

To verify that this is the problem, examine the deployment descriptors for all enterprise bean server applications running in the server in search for a jndiName that is specified in more than one enterprise bean application.

To correct the problem, change any duplicate jndiName values to ensure that each enterprise bean in the server process is bound with a different name.

ConfigurationException from "new InitialContext" operation or from a JNDI Context operation with a URL name

If you are attempting to obtain an initial JNDI context, a configuration exception can occur because an invalid JNDI property value was passed to the InitialContext constructor. This includes JNDI properties set in the System properties or in some jndi.properties file visible to the class loader in effect. A malformed provider URL is the most likely property to be incorrect. If the JNDI client is being run as a thin client such that the CLASSPATH is set to include all of the individual jar files required, make sure the .jar file containing the properties file com/ibm/websphere/naming/jndiprovider.properties is in the CLASSPATH.

If the exception is occurring from a JNDI Context call with a name in the form of a URL, the current JNDI configuration may not be set up properly so that the required factory class name cannot be determined, or the factory may not be visible to the class loader currently in effect. If the name is a Java: URL, the JNDI client must be running in a J2EE client or server environment. That is, the client must be running in a container.

Check the exception message to verify the cause.

If the exception is being thrown from the InitialContext constructor, correct the property setting or the CLASSPATH.

If the exception is being thrown from a JNDI Context method, make sure the property java.naming.factory.url.pkgs includes the package name for the factory required for the URL scheme in

the name. URL names with the Java scheme can only be used while running in a container.

ServiceUnavailableException from ″new InitialContext″ operation

This exception indicates that some unexpected problem occurred while attempting to contact the name server to obtain an initial context. The ServiceUnavailableException, like all NamingException objects, can be queried for a root cause. Check the root cause for more information. It is possible that some of the problems described for CommunicationExceptions may also result in a ServiceUnavailableException.

Since this exception is triggered by an unexpected error, there is no probable cause to confirm. If the root cause exception does not indicate what the probable cause is, investigate the possible causes listed for CommunicationExceptions.

CommunicationException thrown from a ″new InitialContext″ operation

The name server identified by the provider URL cannot be contacted to obtain the initial JNDI context. There are many possible causes for this problem, including:
- The host name or port in the provider URL is incorrect.
- The host name cannot be resolved into an IP address by the domain name server, or the IP address does not match the IP address which the server is actually running under.
- A firewall on the client or server is preventing the port specified in the provider URL from being used.

To correct this problem:
- Make sure the provider URL and the network configurations on the client and server machines are correct.
- Make sure the host name can be resolved into an IP address which can be reached by the client machine. You can do this using the ping command.
- If you are running a firewall, make sure that use of the port specified in the provider URL will be allowed.

dumpNameSpace tool

You can use the dumpNameSpace tool to dump the contents of a name space accessed through a name server. The dumpNameSpace tool is based on Java Naming and Directory Interface (JNDI).

When you run the dumpNameSpace tool, the naming service must be active. The dumpNameSpace tool cannot dump name spaces local to the server process, such as those with java: and local: URL schemes. The local: name space contains references to enterprise beans with local interfaces. Use the name space dump utility for java:, local: and server name spaces to dump java: and local: name spaces.

The tool dumps the server root context for the server at the specified host and port unless you specify a non-default starting context which precludes it. The tool does not dump the server root contexts for other servers.

Running dumpNameSpace

You can run the tool from a command line or using its program interface. This topic describes command-line invocations. To access the dumpNameSpace tool through its program interface, refer to the class `com.ibm.websphere.naming.DumpNameSpace` in the WebSphere Application Server API documentation. “Example: Invoking the name space dump tool” on page 369 illustrates running the tool from a command line or using its program interface.

To run the tool from a command line, enter the following command from the *WebSphere/AppServer/bin* directory:

Platform	Command
UNIX	`dumpNameSpace.sh [[-keyword value]...]`
Windows NT or later	`dumpNameSpace [[-keyword value]...]`

If you run the dumpNameSpace tool with security enabled, a login prompt is displayed. If you cancel the login prompt, the dumpNameSpace tool continues outbound with an ″UNAUTHENTICATED″ credential. Thus, by default, an ″UNAUTHENTICATED″ credential is used that is equivalent to the ″Everyone″ access authorization policy. You can modify this default setting by changing the value for the `com.ibm.CSI.performClientAuthenticationRequired` property to `true` in the *`install_dir`*`/properties/sas.client.props` file. If you change this property to `true`, rerun the dumpNameSpace tool and cancel the login prompt; the authorization fails and the command does not continue outbound.

Parameters

The keywords and associated values for the dumpNameSpace tool follow:

-host *myhost.company.com*
Indicates the bootstrap host or the WebSphere Application Server host whose name space you want to dump. The value defaults to `localhost`. Specify a value for `-host` if the tool is not run from the local machine. The `-host` parameter instructs the tool to connect to a server on a remote machine. For example, run

```
dumpNameSpace -host myhost.mycompany.com
```

to display the name space of the server running on *myhost.mycompany.com*.

-port *nnn*
Indicates the bootstrap port which, if not specified, defaults to `2809`.

-root { cell | server | node | host | legacy | tree | default }
Indicates the root context to use as the initial context for the dump. The applicable root options and default root context depend on the type of name server from which the dump is being obtained. Descriptions of `-root` options follow.

For WebSphere Application Server Version 5.0 or later servers:

cell	DumpNameSpace default for product Version 5.0 or later servers. Dumps the tree starting at the cell root context.
server	Dumps the tree starting at the server root context.
node	Dumps the tree starting at the node root context.
tree	Dumps the tree starting at the tree root context.

For all WebSphere Application Server and other name servers:

default	Dumps the tree starting at the initial context which JNDI returns by default for that server type. This is the only `-root` option that is compatible with non-WebSphere Application Server name servers.

-url *some_provider_URL*
Indicates the value for the java.naming.provider.url property used to get the initial JNDI context. This option can be used in place of the `-host`, `-port`, and `-root` options. If the `-url` option is specified, the `-host`, `-port`, and `-root` options are ignored.

-factory *com.ibm.websphere.naming.WsnInitialContextFactory*
Indicates the initial context factory to be used to get the JNDI initial context. The value defaults to *com.ibm.websphere.naming.WsnInitialContextFactory*. The default value generally does not need to be changed.

-startAt *some/subcontext/in/the/tree*
Indicates the path from the bootstrap host's root context to the top level context where the dump should begin. The tool recursively dumps subcontexts below this point. It defaults to an empty string, that is, the bootstrap host root context.

-format { jndi | ins }

jndi	The default. Displays name components as atomic strings.
ins	Shows name components parsed using Interoperable Naming Service (INS) rules (`id.kind`).

-report { short | long }

short	The default. Dumps the binding name and bound object type. This output is also provided by JNDI Context.list().
long	Dumps the binding name, bound object type, local object type, and string representation of the local object (that is, the IORs, string values, and other values that are printed). For objects of user-defined classes to display correctly with the long report option, you might need to add their containing directories to the list of directories searched. Set the environment variable **WAS_USER_DIRS** as shown in the following platform-specific commands. The value can include one or more directories. **UNIX** `WAS_USER_DIRS=/usr/classdir1:/usr/classdir2 export WAS_USER_DIRS` **Windows NT or later** set `WAS_USER_DIRS=c:\classdir1;d:\classdir2` All `.zip`, `.jar`, and `.class` files in the specified directories can then be resolved by the class loader when running the dumpNameSpace tool.

-traceString *"some.package.name.to.trace.*=all=enabled"*
Represents the trace string with the same format as that generated by the servers. The output is sent to the file `DumpNameSpaceTrace.out`.

Example: Invoking the name space dump tool

It is often helpful to view a dump of the name space to understand why a naming operation is failing. You can invoke the name space dump tool from the command line or from a program. Examples of each option follow.

Invoking the name space dump tool from a command line

Invoke the name space dump tool from a command line by entering either of the following commands:

```
dumpNameSpace -host myhost.mycompany.com -port 901
```

or:

```
dumpNameSpace -url corbaloc:iiop:myhost.mycompany.com:901
```

There are several command-line options to choose from. For detailed help on the options, enter either of the following commands:

```
dumpNameSpace -help
```

or:

```
dumpNameSpace -?
```

Invoking the name space dump tool from a Java program

You can dump name spaces from a program with the com.ibm.websphere.naming.DumpNameSpace API. Refer to the WebSphere Application Server API documentation for details on the DumpNameSpace program interface.

The following example illustrates how to invoke the name space dump tool from a Java program:

```
{
   ...
   import javax.naming.Context;
   import javax.naming.InitialContext;
   import com.ibm.websphere.naming.DumpNameSpace;
   ...
   java.io.PrintStream filePrintStream = ...
   Context ctx = new InitialContext();
   // Starting context for dump
   ctx = (Context) ctx.lookup("cell/nodes/node1/servers/server1");
   DumpNameSpace dumpUtil =
      new DumpNameSpace(filePrintStream, DumpNameSpace.SHORT);
   dumpUtil.generateDump(ctx);
   ...
}
```

Name space dump utility for `java:`, `local:` and `server` name spaces

Sometimes it is helpful to dump the `java:` name space for a J2EE application. You cannot use the dumpNameSpace command line utility for this purpose because the application's `java:` name space is accessible only by that J2EE application. From the WebSphere Application Server scripting tool, you can invoke a NameServer MBean to dump the `java:` name space for any J2EE application running in that same server process.

There is another name space local to server process which you cannot dump with the dumpNameSpace command line utility. This name space has the URL scheme of local: and is used by the container to bind objects locally instead of through the name server. The `local:` name space contains references to enterprise beans with local interfaces. There is only one `local:` name space in a server process. You can dump the `local:` name space by invoking the NameServer MBean associated with that server process.

Name space dump options

Name space dump options are specified in the MBean invocation as a parameter in character string format. The option descriptions follow.

-startAt *some/subcontext/in/the/tree*

Indicates the path from the name space root context to the top level context where the dump should begin. The utility recursively dumps subcontexts below this point. It defaults to an empty string, that is, the root context.

-report {short | long}

Option	Description
short	The default. Dumps the binding name and bound object type. This output is also provided by JNDI Context.list().
long	Dumps the binding name, bound object type, local object type, and string representation of the local object (that is, the IORs, string values, and other values that are printed).

-root {tree | host | legacy | cell | node | server | default}

Specify the root context of where the dump should start. The default value for `-root` is *cell*. This option is only valid for server name space dumps.

Option	Description
tree	Dump the tree starting at the tree root context.
host	Dump the tree starting at the server host root context (synonymous with ″node″).
legacy	Dump the tree starting at the legacy root context.
cell	Dump the tree starting at the cell root context. This is the default option.
node	Dump the tree starting at the node root context (synonymous with ″host″).
server	Dump the tree starting at the server root context. This is `-root` default.
default	Dump the tree starting at the initial context which JNDI returns by default for that server type.

-format {jndi | ins}

Specify the format to display name component as atomic strings or parsed according to INS rules (id.kind). This option is only valid for server name space dumps.

Option	Description
jndi	Display name components as atomic strings. This is `-format` default.
ins	Display name components parsed according to INS rules (id.kind).

NameServer MBean invocation

1. Enter the WebSphere Application Server scripting command prompt.

   Invoke a method on a NameServer MBean by using the WebSphere Application Server scripting tool. Enter the scripting command prompt by typing the following command:

Platform	Command
UNIX	wsadmin.sh
Windows NT	wsadmin

   Use the `-help` option for help on using the wsadmin command.

2. Select the NameServer MBean instance to invoke.

   Execute the following script commands to select the NameServer instance you want to invoke. For example,

   ```
   set mbean [$AdminControl completeObjectName WebSphere:*,type=NameServer,cell=
       cellName,node=nodeName,process=serverName]
   ```

   where *cellName*, *nodeName*, and *serverName* are the names of the cell, node, and server for the MBean you want to invoke. The specified server must be running before you can invoke a method on the MBean.

   You can see a list of all NameServer MBeans current running by issuing the following query:

   ```
   $AdminControl queryNames {*:*,type=NameServer}
   ```

3. Invoke the NameServer MBean.

   java: name space

   Dump a java: name space by invoking the dumpJavaNameSpace method on the NameServer MBean. Since each server application has its own `java:` name space, the application must be specified on the method invocation. An application is identified by the application name, module name, and component name. The method syntax follows:

```
$AdminControl invoke $mbean dumpJavaNameSpace {{appname}{modName}{compName}{opts}}
```

where *appName* is the application name, *modName* is the module name, and *compName* is the component name of the `java:` name space you want to dump. The value for *opts* is the list of name space dump options described earlier in this section. The list can be empty.

local: name space

Dump a `java:` name space by invoking the dumpLocalNameSpace method on the NameServer MBean. Because there is only one `local:` name space in a server process, you have to specify the name space dump options only.

```
$AdminControl invoke $mbean dumpLocalNameSpace {{opts}}
```

where *opts* is the list of name space dump options described earlier in this section. The list can be empty.

Server name space

Dump a server name space by invoking the dumpServerNameSpace method on an application server's NameServer MBean. This provides an alternative way to dump the name space on an application server, much like the dumpNameSpace command line utility.

```
$AdminControl invoke $mbean dumpServerNameSpace {{opts}}
```

where *opts* is the list of name space dump options described earlier in this section. The list can be empty.

Name space dump output

Name space dump output is sent to the console. It is also written to the file `DumpNameSpace.log`, in the server's log directory.

Example: Invoking the name space dump utility for `java:` and `local:` name spaces

It is often helpful to view the dump of a `java:` or `local:` name space to understand why a naming operation is failing. The NameServer MBean running in the application's server process can be invoked from the WebSphere Application Server scripting tool to generate a dump of these name spaces. Examples of NameServer MBean calls to generate dumps of java: and local: name spaces follow.

Dumping a `java:` name space

Assume you want to dump the `java:` name space of an application component running in server server1 on node node1 of the cell MyCell. The application name is AcctApp in module AcctApp.war, and the component name is Acct Servlet. The following script commands generate a long format dump of the application's `java:` name space of that application:

```
set mbean [$AdminControl completeObjectName WebSphere:*,type=NameServer,cell=MyCell,node=node1,process=server1]
$AdminControl invoke $mbean dumpJavaNameSpace {{DefaultApplication}{Increment.jar}{Increment}{-report long}}
```

Dumping a `local:` name space

Assume you want to dump the `local:` name space for the server server1 on node node1 of cell MyCell. The following script commands will generate a short format dump of that server's local name space:

```
set mbean [$AdminControl completeObjectName WebSphere:*type=NameServer,cell=MyCell,node=node1,process=server1]
$AdminControl invoke $mbean dumpLocalNameSpace {{-report short}}
```

Name space dump sample output

Name space dump output is available in short or long format.

Name space dump output looks like the following example, which is the **SHORT** dump format:

```
Getting the initial context
Getting the starting context

================================================================================
Name Space Dump
   Provider URL: corbaloc:iiop:localhost:9810
   Context factory: com.ibm.websphere.naming.WsnInitialContextFactory
   Requested root context: cell
   Starting context: (top)=outpostNetwork
   Formatting rules: jndi
   Time of dump: Mon Sep 16 18:35:03 CDT 2002
================================================================================


================================================================================
Beginning of Name Space Dump
================================================================================

    1 (top)
    2 (top)/domain                                  javax.naming.Context
    2    Linked to context: outpostNetwork
    3 (top)/cells                                   javax.naming.Context
    4 (top)/clusters                                javax.naming.Context
    5 (top)/clusters/Cluster1                       javax.naming.Context
    6 (top)/cellname                                java.lang.String
    7 (top)/cell                                    javax.naming.Context
    7    Linked to context: outpostNetwork
    8 (top)/deploymentManager                       javax.naming.Context
    8    Linked to URL: corbaloc::outpost:9809/NameServiceServerRoot
    9 (top)/nodes                                   javax.naming.Context
   10 (top)/nodes/will2                             javax.naming.Context
   11 (top)/nodes/will2/persistent                  javax.naming.Context
   12 (top)/nodes/will2/persistent/SomeObject       SomeClass
   13 (top)/nodes/will2/nodename                    java.lang.String
   14 (top)/nodes/will2/domain                      javax.naming.Context
   14    Linked to context: outpostNetwork
   15 (top)/nodes/will2/cell                        javax.naming.Context
   15    Linked to context: outpostNetwork
   16 (top)/nodes/will2/servers                     javax.naming.Context
   17 (top)/nodes/will2/servers/server1             javax.naming.Context
   18 (top)/nodes/will2/servers/will2               javax.naming.Context
   19 (top)/nodes/will2/servers/member2             javax.naming.Context
   20 (top)/nodes/will2/node                        javax.naming.Context
   20    Linked to context: outpostNetwork/nodes/will2
   21 (top)/nodes/will2/nodeAgent                   javax.naming.Context
   22 (top)/nodes/outpost                           javax.naming.Context
   23 (top)/nodes/outpost/node                      javax.naming.Context
   23    Linked to context: outpostNetwork/nodes/outpost
   24 (top)/nodes/outpost/nodeAgent                 javax.naming.Context
   24    Linked to URL: corbaloc::outpost:2809/NameServiceServerRoot
   25 (top)/nodes/outpost/persistent                javax.naming.Context
   26 (top)/nodes/outpost/nodename                  java.lang.String
   27 (top)/nodes/outpost/domain                    javax.naming.Context
   27    Linked to context: outpostNetwork
   28 (top)/nodes/outpost/servers                   javax.naming.Context
   29 (top)/nodes/outpost/servers/server1           javax.naming.Context
   30 (top)/nodes/outpost/servers/server1/url       javax.naming.Context
   31 (top)/nodes/outpost/servers/server1/url/CatalogDAOSQLURL
   31                                               java.net.URL
   32 (top)/nodes/outpost/servers/server1/mail      javax.naming.Context
   33 (top)/nodes/outpost/servers/server1/mail/PlantsByWebSphere
```

```
  33                                          javax.mail.Session
  34 (top)/nodes/outpost/servers/server1/TransactionFactory
  34                                          com.ibm.ejs.jts.jts.ControlSet$LocalFactory
  35 (top)/nodes/outpost/servers/server1/servername java.lang.String
  36 (top)/nodes/outpost/servers/server1/WSsamples  javax.naming.Context
  37 (top)/nodes/outpost/servers/server1/WSsamples/TechSampDatasource
  37                                          TechSamp
  38 (top)/nodes/outpost/servers/server1/thisNode   javax.naming.Context
  38    Linked to context: outpostNetwork/nodes/outpost
  39 (top)/nodes/outpost/servers/server1/cell       javax.naming.Context
  39    Linked to context: outpostNetwork
  40 (top)/nodes/outpost/servers/server1/eis        javax.naming.Context
  41 (top)/nodes/outpost/servers/server1/eis/DefaultDatasource_CMP
  41                                          Default_CF
  42 (top)/nodes/outpost/servers/server1/eis/WSsamples  javax.naming.Context
  43 (top)/nodes/outpost/servers/server1/eis/WSsamples/TechSampDatasource_CMP
  43                                          TechSamp_CF
  44 (top)/nodes/outpost/servers/server1/eis/jdbc   javax.naming.Context
  45 (top)/nodes/outpost/servers/server1/eis/jdbc/PlantsByWebSphereDataSource_CMP
  45                                          PLANTSDB_CF
  46 (top)/nodes/outpost/servers/server1/eis/jdbc/petstore
  46                                          javax.naming.Context
  47 (top)/nodes/outpost/servers/server1/eis/jdbc/petstore/PetStoreDB_CMP
  47                                          PetStore_CF
  48 (top)/nodes/outpost/servers/server1/eis/jdbc/CatalogDB_CMP
  48                                          Catalog_CF
  49 (top)/nodes/outpost/servers/server1/jta        javax.naming.Context
  50 (top)/nodes/outpost/servers/server1/jta/usertransaction
  50                                          java.lang.Object
  51 (top)/nodes/outpost/servers/server1/DefaultDatasource
  51                                          Default Datasource
  52 (top)/nodes/outpost/servers/server1/jdbc       javax.naming.Context
  53 (top)/nodes/outpost/servers/server1/jdbc/CatalogDB CatalogDB
  54 (top)/nodes/outpost/servers/server1/jdbc/petstore  javax.naming.Context
  55 (top)/nodes/outpost/servers/server1/jdbc/petstore/PetStoreDB
  55                                          PetStoreDB
  56 (top)/nodes/outpost/servers/server1/jdbc/PlantsByWebSphereDataSource
  56                                          PLANTSDB
  57 (top)/nodes/outpost/servers/outpost            javax.naming.Context
  57    Linked to URL: corbaloc::outpost:2809/NameServiceServerRoot
  58 (top)/nodes/outpost/servers/member1            javax.naming.Context
  59 (top)/nodes/outpost/cell                       javax.naming.Context
  59    Linked to context: outpostNetwork
  60 (top)/nodes/outpostManager                     javax.naming.Context
  61 (top)/nodes/outpostManager/domain              javax.naming.Context
  61    Linked to context: outpostNetwork
  62 (top)/nodes/outpostManager/cell                javax.naming.Context
  62    Linked to context: outpostNetwork
  63 (top)/nodes/outpostManager/servers             javax.naming.Context
  64 (top)/nodes/outpostManager/servers/dmgr        javax.naming.Context
  64    Linked to URL: corbaloc::outpost:9809/NameServiceServerRoot
  65 (top)/nodes/outpostManager/node                javax.naming.Context
  65    Linked to context: outpostNetwork/nodes/outpostManager
  66 (top)/nodes/outpostManager/nodename            java.lang.String
  67 (top)/persistent                               javax.naming.Context
  68 (top)/persistent/cell                          javax.naming.Context
  68    Linked to context: outpostNetwork
  69 (top)/legacyRoot                               javax.naming.Context
  69    Linked to context: outpostNetwork/persistent
  70 (top)/persistent/AnotherObject                 AnotherClass

================================================================================
End of Name Space Dump
================================================================================
```

Chapter 25. Object Request Broker

Object Request Broker communications trace

The Object Request Broker (ORB) communications trace, typically referred to as *CommTrace*, contains the sequence of General InterORB Protocol (GIOP) messages sent and received by the ORB when the application is running.

It might be necessary to understand the low-level sequence of client-to-server or server-to-server interactions during problem determination. This topic uses trace entries from log examples 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 that displays when intervening with the GIOP-message boundaries.

Location

When ORB tracing is enabled, this information is placed in the *app_server_root*/profiles/*profile_name*/logs/*server_name*/trace.log directory.

About the ORB trace file

The following are properties of the file that is created when ORB tracing is enabled.
- Read-only
- Updated by the administrative function
- Use this file to localize and resolve ORB-related problems.

How to interpret the output

The following sections refer to sample log output found later in this topic.

Identifying information

The start of a GIOP message is identified by a line that contains either OUT GOING: or IN COMING: depending on whether the message is sent or received by the process.

Following the identifying line entry is a series of items, formatted for convenience, with information extracted from the raw message that identifies the endpoints in this particular message interaction. See lines 3-13 in both examples. The formatted items include the following:
- GIOP message type (line 3)
- Date and time that the message was recorded (line 4)
- Information that is useful to identify the thread that is running when the message records, and other thread-specific information (line 5)
- Local and remote TCP/IP ports used for the interaction (lines 6 through 9)
- GIOP version, byte order, an indication of whether the message is a fragment, and message size (lines 10 through 13)

Request ID, response expected and reply status

Following the introductory message information, the request ID is an integer generated by the ORB. It is used to identify and associate each request with its corresponding reply. This association 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 15-17 in the request example show the request ID, followed by an indication to the receiving endpoint that a response is expected (CORBA supports sending one-way requests for which a response is not expected.)
- Line 15 in the *Sample Log Entry - GIOP Reply* shows the request ID; line 33 shows the reply status received after completing the previously sent request.

Object Key

Lines 18-20 in the request example show the object key, the internal representation used by the ORB to identify and locate the target object intended to receive the request message. Object keys are not standardized.

Operation

Line 21 in the request example shows the name of the operation that the target object starts in the receiving endpoint. In this example, the specific operation requested is named _get_value.

Service context information

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

Lines 22-41 in the request example illustrate typical service context entries. Three service contexts are in the request message, as shown in line 22. The ID, length of data, and raw data for each service context is printed next. Lines 23-25 show an IBM-proprietary context, as indicated by the 0x49424D12 ID. Lines 26-41 show two standard service contexts, identified by 0x6 ID (line 26) and the 0x1 ID (line 39).

Lines 16-32 in the *Sample Log Entry - GIOP Reply* illustrate two service contexts, one IBM-proprietary (line 17) and one standardized (line 20).

For the definition of the standardized service contexts, see the CORBA specification. Service context 0x1 (CORBA::IOP::CodeSets) is used to publish the character code sets supported by the ORB in order to negotiate and determine the code set used to transmit character data. Service context 0x6 (CORBA::IOP::SendingContextRunTime) is used by Remote Method Invocation over the Internet Inter-ORB Protocol (RMI-IIOP) to provide the receiving endpoint with the IOR for the SendingContextRuntime object. IBM service context 0x49424D12 is used to publish ORB PartnerVersion information to support release-to-release interoperability between sending and receiving ORBs.

Data offset

Line 42 in the request example shows 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 45 in the request example and line 36 in the *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. For brevity, not all of the raw data is in the figures.

Sample Log Entry - GIOP Request

```
1.  OUT GOING:

3.  Request Message
4.  Date:          April 17, 2002 10:00:43 PM CDT
5.  Thread Info:   P=842115:O=1:CT
6.  Local Port:    1243 (0x4DB)
7.  Local IP:      jdoe.austin.ibm.com/192.168.1.101
8.  Remote Port:   1242 (0x4DA)
9.  Remote IP:     jdoe.austin.ibm.com/192.168.1.101
10.  GIOP Version:  1.2
11. Byte order:    big endian
12. Fragment to follow: No
```

```
13. Message size:  268 (0x10C)
--
15. Request ID:        5
16. Response Flag:     WITH_TARGET
17. Target Address:     0
18. Object Key:        length = 24 (0x18)
                       4B4D4249 00000010 BA4D6D34 000E0008
                       00000000 00000000
21. Operation:         _get_value
22. Service Context:   length = 3 (0x3)
23. Context ID:  1229081874 (0x49424D12)
24. Context data:  length = 8 (0x8)
                       00000000 13100003
26. Context ID:  6 (0x6)
27. Context data:  length = 164 (0xA4)
                       00000000 00000028 49444C3A 6F6D672E
                       6F72672F 53656E64 696E6743 6F6E7465
                       78742F43 6F646542 6173653A 312E3000
                       00000001 00000000 00000068 00010200
                       0000000E 3139322E 3136382E 312E3130
                       310004DC 00000018 4B4D4249 00000010
                       BA4D6D69 000E0008 00000000 00000000
                       00000002 00000001 00000018 00000000
                       00010001 00000001 00010020 00010100
                       00000000 49424D0A 00000008 00000000
                       13100003
39. Context ID:  1 (0x1)
40. Context data:  length = 12 (0xC)
                       00000000 00010001 00010100
42. Data Offset:       118


45. 0000: 47494F50 01020000 0000010C 00000005   GIOP............
46. 0010: 03000000 00000000 00000018 4B4D4249   ............KMBI
47. 0020: [remainder of message body deleted for brevity]
```

Sample Log Entry - GIOP Reply

```
1.  IN COMING:

3.  Reply Message
4.  Date:          April 17, 2002 10:00:47 PM CDT
5.  Thread Info:   RT=0:P=842115:O=1:com.ibm.rmi.transport.TCPTransportConnection
5a (line 5 broken for publication).   remoteHost=192.168.1.101 remotePort=1242 localPort=1243
6.  Local Port:    1243 (0x4DB)
7.  Local IP:      jdoe.austin.ibm.com/192.168.1.101
8.  Remote Port:   1242 (0x4DA)
9.  Remote IP:     jdoe.austin.ibm.com/192.168.1.101
10. GIOP Version:  1.2
11. Byte order:    big endian
12. Fragment to follow: No
13. Message size:  208 (0xD0)
--
15. Request ID:        5
16. Service Context:   length = 2 (0x2)
17. Context ID:  1229081874 (0x49424D12)
18. Context data:  length = 8 (0x8)
                       00000000 13100003
20. Context ID:  6 (0x6)
21. Context data:  length = 164 (0xA4)
                       00000000 00000028 49444C3A 6F6D672E
                       6F72672F 53656E64 696E6743 6F6E7465
                       78742F43 6F646542 6173653A 312E3000
                       00000001 00000000 00000068 00010200
                       0000000E 3139322E 3136382E 312E3130
                       310004DA 00000018 4B4D4249 00000010
                       BA4D6D34 000E0008 00000001 00000000
```

```
                      00000002 00000001 00000018 00000000
                      00010001 00000001 00010020 00010100
                      00000000 49424D0A 00000008 00000000
                      13100003
33. Reply Status:     NO_EXCEPTION


36. 0000: 47494F50 01020001 000000D0 00000005   GIOP............
37. 0010: 00000000 00000002 49424D12 00000008   ........IBM.....
38. 0020: [remainder of message body deleted for brevity]
```

Object request broker troubleshooting tips

Use these tips to diagnose problems related to the WebSphere Application Server Object Request Broker (ORB).

Enabling tracing for the Object Request Broker component

The object request broker (ORB) service is one of the WebSphere Application Server run time services. Tracing messages sent and received by the ORB is a useful starting point for troubleshooting the ORB service. You can selectively enable or disable tracing of ORB messages for each server in a WebSphere Application Server installation, and for each application client.

This tracing is referred to by WebSphere Application Server support as a *comm trace*, and is different from the general purpose trace facility. The trace facility, which shows the detailed run-time behavior of product components, may be used alongside comm trace for other product components, or for the ORB component. The trace string associated with the ORB service is `ORBRas=all=enabled`.

You can enable and disable comm tracing using the administrative console or by manually editing the server.xml file for the server be trace. You must stop and restart the server for the configuration change to take effect.

For example, using the administrative console:

- Click **Servers > Application servers >** *server_name* **> Container services > ORB service**, and select the ORB tracing. Click **OK**, and then click **Save** to save your settings. Restart the server for the new settings to take effect. Or,
- Locate the server.xml file for the selected server, for example: *profile_root*/config/cells/*node_name*/nodes/*node_name*/servers/*server_name*/server.xml.
- Locate the services entry for the ORB service, `xmi:type=orb:ObjectRequestBroker`, and set `commTraceEnabled=true`.

ORB tracing for client applications requires that both the ORBRas component trace and the ORB comm trace are enabled. If only the ORB comm trace is enabled, no trace output is generated for client-side ORB traces. You can use one of the following options to enable both traces in the command line used to launch the client application:

- If you are using the WebSphere Application Server launcher, launchClient, use the **-CCD** option.
- If you are using the **java** command specify these parameters:

```
-trace=ORBRas=all=enabled
-tracefile=filename
-Dcom.ibm.CORBA.Debug=true
-Dcom.ibm.CORBA.CommTrace=true
```

ORB tracing output for thin clients can be directed by setting the `com.ibm.CORBA.Debug.Output = debugOutputFilename` parameter in the command line.

Important: When using launchClient on operating systems like AIX or Linux, because ORB tracing is integrated with the WebSphere Application Server general component trace, both the component and comm ORB traces are written to the same file as the rest of the WebSphere Application Server traces. The name of this file is specified on the `-CCtracefile=`*filename* parameter.

When using launchClient on a Windows operating systems, you get a separate ORB trace file, called orbtrc.*timestamp*.txt. This file is located in the current directory.

Log files and messages associated with Object Request Broker

Messages and trace information for the ORB are saved in the following logs:
- The *profile_root*/logs/*server_name*/trace.log file for output from communications tracing and tracing the behavior of the ORBRas component
- The JVM logs for each application server, for WebSphere Application Server error and warning messages

The following message in the SystemOut.log file indicates the successful start of the application server and its ORB service:

WSVR0001I: Server server1 open for e-business

When communications tracing is enabled, a message similar to the following example in the *profile_root*/logs/*server_name*/trace.log file, indicates that the ORB service has started successfully. The message also shows the start of a listener thread, which is waiting for requests on the specified local port.

com.ibm.ws.orbimpl.transport.WSTransport startListening(ServerConnectionData connectionData) P=693799:O=0:CT a new ListenerThread has been started for ServerSocket[addr=0.0.0.0/ 0.0.0.0,port=0,localport=1360]

When the `com.ibm.ejs.oa.*=all=enabled` parameter is specified, tracing of the Object Adapter is enabled. The following message in the trace.log indicates that the ORB service started successfully:

EJSORBImpl < initializeORB

The ORB service is one of the first services started during the WebSphere Application Server initialization process. If it is not properly configured, other components such as naming, security, and node agent, are not likely to start successfully. This is obvious in the JVM logs or trace.log of the affected application server.

Adjusting object request broker timeout values

If Web clients that access Java applications running in the product environment are consistently experiencing problems with their requests, and the problem cannot be traced to other sources and addressed through other solutions, consider setting an ORB timeout value and adjusting it for your environment. A list of timeout scenarios follows:
- Web browsers vary in their language for indicating that they have timed out. Usually, the problem is announced as a connection failure or a no-path-to-server message.
- Set an ORB timeout value to less than the time after which a Web client eventually times out. Because it can be difficult to tell how long Web clients wait before timing out, setting an ORB timeout value requires experimentation. The ideal testing environment features some simulated network failures for testing the proposed setting value.

- Empirical results from limited testing indicate that 30 seconds is a reasonable starting value. Ensure that this setting is not too low. To fine tune the setting, find a value that is not too low. Gradually decrease the setting until reaching the threshold at which the value becomes too low. Set the value a little higher than the threshold.
- When an ORB timeout value is set too low, the symptom is numerous CORBA NO_RESPONSE exceptions, which occur even for some valid requests. The value is likely to be too low if requests that should have been successful, for example, the server is not down, are being lost or refused.

Timeout adjustments: Do not adjust an ORB timeout value unless you have a problem. Configuring a value that is inappropriate for the environment can create a problem. An incorrect value can produce results worse than the original problem.

You can adjust timeout intervals for the product Java ORB through the following administrative settings:
- **Request timeout**, the number of seconds to wait before timing out on most pending ORB requests if the network fails
- **Locate request timeout**, the number of seconds to wait before timing out on a locate-request message

Read the information about Object Request Broker service settings in the *Setting up the application serving environment* PDF book.

Java packages containing the Object Request Broker service

The ORB service resides in the following Java packages:
- com.ibm.com.CORBA.*
- com.ibm.rmi.*
- com.ibm.ws.orb.*
- com.ibm.ws.orbimpl.*
- org.omg.CORBA.*
- javax.rmi.CORBA.*

JAR files that contain the previously mentioned packages include:
- *app_server_root*/java/jre/lib/ext/ibmorb.jar
- *app_server_root*/java/jre/lib/ext/iwsorbutil.jar
- *app_server_root*/lib/iwsorb.jar

Tools used with Object Request Broker

The tools used to compile Java remote interfaces to generate language bindings used by the ORB at runtime reside in the following Java packages:
- com.ibm.tools.rmic.*
- com.ibm.idl.*

The JAR file that contains the packages is *app_server_root*/java/lib/ibmtools.jar.

Object Request Broker properties

The ORB service requires a number of ORB properties for correct operation. It is not necessary for most users to modify these properties, and it is recommended that only your system administrator modify them when required.. Consult IBM Support personnel for assistance. The properties reside in the orb.properties file, located in *app_server_root*/java/jre/lib/orb.properties.

CORBA minor codes

The CORBA specification defines standard minor exception codes for use by the ORB when a system exception is thrown. In addition, the object management group (OMG) assigns each vendor a unique prefix value for use in vendor-proprietary minor exception codes. Minor code values assigned to IBM and used

by the ORB in the WebSphere Application Server follow. The minor code value is in decimal and hexadecimal formats. The column labeled minor code reason gives a short description of the condition causing the exception. Currently there is no documentation for these errors beyond the minor code reason. If you require technical support from IBM, the minor code helps support engineers determine the source of the problem.

Table 6. Decimal minor exception codes 1229066320 to 1229066364

Decimal	Hexadecimal	Minor code reason
1229066320	0x49421050	HTTP_READER_FAILURE
1229066321	0x49421051	COULD_NOT_INSTANTIATE_CLIENT_SSL_SOCKET_FACTORY
1229066322	0x49421052	COULD_NOT_INSTANTIATE_SERVER_SSL_SOCKET_FACTORY
1229066323	0x49421053	CREATE_LISTENER_FAILED_1
1229066324	0x49421054	CREATE_LISTENER_FAILED_2
1229066325	0x49421055	CREATE_LISTENER_FAILED_3
1229066326	0x49421056	CREATE_LISTENER_FAILED_4
1229066327	0x49421057	CREATE_LISTENER_FAILED_5
1229066328	0x49421058	INVALID_CONNECTION_TYPE
1229066329	0x49421059	HTTPINPUTSTREAM_NO_ACTIVEINPUTSTREAM
1229066330	0x4942105a	HTTPOUTPUTSTREAM_NO_OUTPUTSTREAM
1229066331	0x4942105b	CONNECTIONINTERCEPTOR_INVALID_CLASSNAME
1229066332	0x4942105c	NO_CONNECTIONDATA_IN_CONNECTIONDATACARRIER
1229066333	0x4942105d	CLIENT_CONNECTIONDATA_IS_INVALID_TYPE
1229066334	0x4942105e	SERVER_CONNECTIONDATA_IS_INVALID_TYPE
1229066335	0x4942105f	NO_OVERLAP_OF_ENABLED_AND_DESIRED_CIPHER_SUITES
1229066352	0x49421070	CAUGHT_EXCEPTION_WHILE_CONFIGURING_SSL_CLIENT_SOCKET
1229066353	0x49421071	GETCONNECTION_KEY_RETURNED_FALSE
1229066354	0x49421072	UNABLE_TO_CREATE_SSL_SOCKET
1229066355	0x49421073	SSLSERVERSOCKET_TARGET_SUPPORTS_LESS_THAN_1
1229066356	0x49421074	SSLSERVERSOCKET_TARGET_REQUIRES_LESS_THAN_1
1229066357	0x49421075	SSLSERVERSOCKET_TARGET_LESS_THAN_TARGET_REQUIRES
1229066358	0x49421076	UNABLE_TO_CREATE_SSL_SERVER_SOCKET
1229066359	0x49421077	CAUGHT_EXCEPTION_WHILE_CONFIGURING_SSL_SERVER_SOCKET
1229066360	0x49421078	INVALID_SERVER_CONNECTION_DATA_TYPE
1229066361	0x49421079	GETSERVERCONNECTIONDATA_RETURNED_NULL
1229066362	0x4942107a	GET_SSL_SESSION_RETURNED_NULL
1229066363	0x4942107b	GLOBAL_ORB_EXISTS
1229066364	0x4942107c	CONNECT_TIME_OUT

Table 7. Decimal minor exception codes 1229123841 to 1229124249

Decimal	Hexadecimal	Minor code reason
1229123841	0x4942f101	DSIMETHOD_NOTCALLED
1229123842	0x4942f102	BAD_INV_PARAMS
1229123843	0x4942f103	BAD_INV_RESULT
1229123844	0x4942f104	BAD_INV_CTX
1229123845	0x4942f105	ORB_NOTREADY
1229123879	0x4942f127	PI_NOT_POST_INIT
1229123880	0x4942f128	INVALID_EXTENDED_PI_CALL
1229123969	0x4942f181	BAD_OPERATION_EXTRACT_SHORT
1229123970	0x4942f182	BAD_OPERATION_EXTRACT_LONG
1229123971	0x4942f183	BAD_OPERATION_EXTRACT_USHORT
1229123972	0x4942f184	BAD_OPERATION_EXTRACT_ULONG
1229123973	0x4942f185	BAD_OPERATION_EXTRACT_FLOAT
1229123974	0x4942f186	BAD_OPERATION_EXTRACT_DOUBLE

Table 7. Decimal minor exception codes 1229123841 to 1229124249 (continued)

1229123975	0x4942f187	BAD_OPERATION_EXTRACT_LONGLONG
1229123976	0x4942f188	BAD_OPERATION_EXTRACT_ULONGLONG
1229123977	0x4942f189	BAD_OPERATION_EXTRACT_BOOLEAN
1229123978	0x4942f18a	BAD_OPERATION_EXTRACT_CHAR
1229123979	0x4942f18b	BAD_OPERATION_EXTRACT_OCTET
1229123980	0x4942f18c	BAD_OPERATION_EXTRACT_WCHAR
1229123981	0x4942f18d	BAD_OPERATION_EXTRACT_STRING
1229123982	0x4942f18e	BAD_OPERATION_EXTRACT_WSTRING
1229123983	0x4942f18f	BAD_OPERATION_EXTRACT_ANY
1229123984	0x4942f190	BAD_OPERATION_INSERT_OBJECT_1
1229123985	0x4942f191	BAD_OPERATION_INSERT_OBJECT_2
1229123986	0x4942f192	BAD_OPERATION_EXTRACT_OBJECT_1
1229123987	0x4942f193	BAD_OPERATION_EXTRACT_OBJECT_2
1229123988	0x4942f194	BAD_OPERATION_EXTRACT_TYPECODE
1229123989	0x4942f195	BAD_OPERATION_EXTRACT_PRINCIPAL
1229123990	0x4942f196	BAD_OPERATION_EXTRACT_VALUE
1229123991	0x4942f197	BAD_OPERATION_GET_PRIMITIVE_TC_1
1229123992	0x4942f198	BAD_OPERATION_GET_PRIMITIVE_TC_2
1229123993	0x4942f199	BAD_OPERATION_INVOKE_NULL_PARAM_1
1229123994	0x4942f19a	BAD_OPERATION_INVOKE_NULL_PARAM_2
1229123995	0x4942f19b	BAD_OPERATION_INVOKE_DEFAULT_1
1229123996	0x4942f19c	BAD_OPERATION_INVOKE_DEFAULT_2
1229123997	0x4942f19d	BAD_OPERATION_UNKNOWN_BOOTSTRAP_METHOD
1229123998	0x4942f19e	BAD_OPERATION_EMPTY_ANY
1229123999	0x4942f19f	BAD_OPERATION_STUB_DISCONNECTED
1229124000	0x4942f1a0	BAD_OPERATION_TIE_DISCONNECTED
1229124001	0x4942f1a1	BAD_OPERATION_DELEGATE_DISCONNECTED
1229124097	0x4942f201	NULL_PARAM_1
1229124098	0x4942f202	NULL_PARAM_2
1229124099	0x4942f203	NULL_PARAM_3
1229124100	0x4942f204	NULL_PARAM_4
1229124101	0x4942f205	NULL_PARAM_5
1229124102	0x4942f206	NULL_PARAM_6
1229124103	0x4942f207	NULL_PARAM_7
1229124104	0x4942f208	NULL_PARAM_8
1229124105	0x4942f209	NULL_PARAM_9
1229124106	0x4942f20a	NULL_PARAM_10
1229124107	0x4942f20b	NULL_PARAM_11
1229124108	0x4942f20c	NULL_PARAM_12
1229124109	0x4942f20d	NULL_PARAM_13
1229124110	0x4942f20e	NULL_PARAM_14
1229124111	0x4942f20f	NULL_PARAM_15
1229124112	0x4942f210	NULL_PARAM_16
1229124113	0x4942f211	NULL_PARAM_17
1229124114	0x4942f212	NULL_PARAM_18
1229124115	0x4942f213	NULL_PARAM_19
1229124116	0x4942f214	NULL_PARAM_20
1229124117	0x4942f215	NULL_IOR_OBJECT
1229124118	0x4942f216	NULL_PI_NAME
1229124119	0x4942f217	NULL_SC_DATA
1229124126	0x4942f21e	BAD_SERVANT_TYPE

Table 7. Decimal minor exception codes 1229123841 to 1229124249 (continued)

1229124127	0x4942f21f	BAD_EXCEPTION
1229124128	0x4942f220	BAD_MODIFIER_LIST
1229124129	0x4942f221	NULL_PROP_MGR
1229124130	0x4942f222	INVALID_PROPERTY
1229124131	0x4942f223	ORBINITREF_FORMAT
1229124132	0x4942f224	ORBINITREF_MISSING_OBJECTURL
1229124133	0x4942f225	ORBDEFAULTINITREF_FORMAT
1229124134	0x4942f226	ORBDEFAULTINITREF_VALUE
1229124135	0x4942f227	OBJECTKEY_SERVERUUID_LENGTH
1229124136	0x4942f228	OBJECTKEY_SERVERUUID_NULL
1229124137	0x4942f229	BAD_REPOSITORY_ID
1229124138	0x4942f22a	BAD_PARAM_LOCAL_OBJECT
1229124139	0x4942f22b	NULL_OBJECT_IOR
1229124140	0x4942f22c	WRONG_ORB
1229124141	0x4942f22d	NULL_OBJECT_KEY
1229124142	0x4942f22e	NULL_OBJECT_URL
1229124143	0x4942f22f	NOT_A_NAMING_CONTEXT
1229124225	0x4942f281	TYPECODEIMPL_CTOR_MISUSE_1
1229124226	0x4942f282	TYPECODEIMPL_CTOR_MISUSE_2
1229124227	0x4942f283	TYPECODEIMPL_NULL_INDIRECTTYPE
1229124228	0x4942f284	TYPECODEIMPL_RECURSIVE_TYPECODES
1229124235	0x4942f28b	TYPECODEIMPL_KIND_INVALID_1
1229124236	0x4942f28c	TYPECODEIMPL_KIND_INVALID_2
1229124237	0x4942f28d	TYPECODEIMPL_NATIVE_1
1229124238	0x4942f28e	TYPECODEIMPL_NATIVE_2
1229124239	0x4942f28f	TYPECODEIMPL_NATIVE_3
1229124240	0x4942f290	TYPECODEIMPL_KIND_INDIRECT_1
1229124241	0x4942f291	TYPECODEIMPL_KIND_INDIRECT_2
1229124242	0x4942f292	TYPECODEIMPL_NULL_TYPECODE
1229124243	0x4942f293	TYPECODEIMPL_BODY_OF_TYPECODE
1229124244	0x4942f294	TYPECODEIMPL_KIND_RECURSIVE_1
1229124245	0x4942f295	TYPECODEIMPL_COMPLEX_DEFAULT_1
1229124246	0x4942f296	TYPECODEIMPL_COMPLEX_DEFAULT_2
1229124247	0x4942f297	TYPECODEIMPL_INDIRECTION
1229124248	0x4942f298	TYPECODEIMPL_SIMPLE_DEFAULT
1229124249	0x4942f299	TYPECODEIMPL_NOT_CDROS

Table 8. Decimal minor exception codes 1229124250 to 1229125764

Decimal	Hexadecimal	Minor code reason
1229124250	0x4942f29a	TYPECODEIMPL_NO_IMPLEMENT_1
1229124251	0x4942f29b	TYPECODEIMPL_NO_IMPLEMENT_2
1229124357	0x4942f305	CONN_PURGE_REBIND
1229124358	0x4942f306	CONN_PURGE_ABORT
1229124359	0x4942f307	CONN_NOT_ESTABLISH
1229124360	0x4942f308	CONN_CLOSE_REBIND
1229124368	0x4942f310	WRITE_ERROR_SEND
1229124376	0x4942f318	GET_PROPERTIES_ERROR
1229124384	0x4942f320	BOOTSTRAP_SERVER_NOT_AVAIL
1229124392	0x4942f328	INVOKE_ERROR
1229124481	0x4942f381	BAD_HEX_DIGIT
1229124482	0x4942f382	BAD_STRINGIFIED_IOR_LEN

Table 8. Decimal minor exception codes 1229124250 to 1229125764 (continued)

1229124483	0x4942f383	BAD_STRINGIFIED_IOR
1229124485	0x4942f385	BAD_MODIFIER_1
1229124486	0x4942f386	BAD_MODIFIER_2
1229124488	0x4942f388	CODESET_INCOMPATIBLE
1229124490	0x4942f38a	LONG_DOUBLE_NOT_IMPLEMENTED_1
1229124491	0x4942f38b	LONG_DOUBLE_NOT_IMPLEMENTED_2
1229124492	0x4942f38c	LONG_DOUBLE_NOT_IMPLEMENTED_3
1229124496	0x4942f390	COMPLEX_TYPES_NOT_IMPLEMENTED
1229124497	0x4942f391	VALUE_BOX_NOT_IMPLEMENTED
1229124498	0x4942f392	NULL_STRINGIFIED_IOR
1229124865	0x4942f501	TRANS_NS_CANNOT_CREATE_INITIAL_NC_SYS
1229124866	0x4942f502	TRANS_NS_CANNOT_CREATE_INITIAL_NC
1229124867	0x4942f503	GLOBAL_ORB_EXISTS
1229124868	0x4942f504	PLUGINS_ERROR
1229124869	0x4942f505	INCOMPATIBLE_JDK_VERSION
1229124993	0x4942f581	BAD_REPLYSTATUS
1229124994	0x4942f582	PEEKSTRING_FAILED
1229124995	0x4942f583	GET_LOCAL_HOST_FAILED
1229124996	0x4942f584	CREATE_LISTENER_FAILED
1229124997	0x4942f585	BAD_LOCATE_REQUEST_STATUS
1229124998	0x4942f586	STRINGIFY_WRITE_ERROR
1229125000	0x4942f588	BAD_GIOP_REQUEST_TYPE_1
1229125001	0x4942f589	BAD_GIOP_REQUEST_TYPE_2
1229125002	0x4942f58a	BAD_GIOP_REQUEST_TYPE_3
1229125003	0x4942f58b	BAD_GIOP_REQUEST_TYPE_4
1229125005	0x4942f58d	NULL_ORB_REFERENCE
1229125006	0x4942f58e	NULL_NAME_REFERENCE
1229125008	0x4942f590	ERROR_UNMARSHALING_USEREXC
1229125009	0x4942f591	SUBCONTRACTREGISTRY_ERROR
1229125010	0x4942f592	LOCATIONFORWARD_ERROR
1229125011	0x4942f593	BAD_READER_THREAD
1229125013	0x4942f595	BAD_REQUEST_ID
1229125014	0x4942f596	BAD_SYSTEMEXCEPTION
1229125015	0x4942f597	BAD_COMPLETION_STATUS
1229125016	0x4942f598	INITIAL_REF_ERROR
1229125017	0x4942f599	NO_CODEC_FACTORY
1229125018	0x4942f59a	BAD_SUBCONTRACT_ID
1229125019	0x4942f59b	BAD_SYSTEMEXCEPTION_2
1229125020	0x4942f59c	NOT_PRIMITIVE_TYPECODE
1229125021	0x4942f59d	BAD_SUBCONTRACT_ID_2
1229125022	0x4942f59e	BAD_SUBCONTRACT_TYPE
1229125023	0x4942f59f	NAMING_CTX_REBIND_ALREADY_BOUND
1229125024	0x4942f5a0	NAMING_CTX_REBINDCTX_ALREADY_BOUND
1229125025	0x4942f5a1	NAMING_CTX_BAD_BINDINGTYPE
1229125026	0x4942f5a2	NAMING_CTX_RESOLVE_CANNOT_NARROW_TO_CTX
1229125032	0x4942f5a8	TRANS_NC_BIND_ALREADY_BOUND
1229125033	0x4942f5a9	TRANS_NC_LIST_GOT_EXC
1229125034	0x4942f5aa	TRANS_NC_NEWCTX_GOT_EXC
1229125035	0x4942f5ab	TRANS_NC_DESTROY_GOT_EXC
1229125036	0x4942f5ac	TRANS_BI_DESTROY_GOT_EXC
1229125042	0x4942f5b2	INVALID_CHAR_CODESET_1

Table 8. Decimal minor exception codes 1229124250 to 1229125764 (continued)

1229125043	0x4942f5b3	INVALID_CHAR_CODESET_2
1229125044	0x4942f5b4	INVALID_WCHAR_CODESET_1
1229125045	0x4942f5b5	INVALID_WCHAR_CODESET_2
1229125046	0x4942f5b6	GET_HOST_ADDR_FAILED
1229125047	0x4942f5b7	REACHED_UNREACHABLE_PATH
1229125048	0x4942f5b8	PROFILE_CLONE_FAILED
1229125049	0x4942f5b9	INVALID_LOCATE_REQUEST_STATUS
1229125050	0x4942f5ba	NO_UNSAFE_CLASS
1229125051	0x4942f5bb	REQUEST_WITHOUT_CONNECTION_1
1229125052	0x4942f5bc	REQUEST_WITHOUT_CONNECTION_2
1229125053	0x4942f5bd	REQUEST_WITHOUT_CONNECTION_3
1229125054	0x4942f5be	REQUEST_WITHOUT_CONNECTION_4
1229125055	0x4942f5bf	REQUEST_WITHOUT_CONNECTION_5
1229125056	0x4942f5c0	NOT_USED_1
1229125057	0x4942f5c1	NOT_USED_2
1229125058	0x4942f5c2	NOT_USED_3
1229125059	0x4942f5c3	NOT_USED_4
1229125512	0x4942f788	BAD_CODE_SET
1229125520	0x4942f790	INV_RMI_STUB
1229125521	0x4942f791	INV_LOAD_STUB
1229125522	0x4942f792	INV_OBJ_IMPLEMENTATION
1229125523	0x4942f793	OBJECTKEY_NOMAGIC
1229125524	0x4942f794	OBJECTKEY_NOSCID
1229125525	0x4942f795	OBJECTKEY_NOSERVERID
1229125526	0x4942f796	OBJECTKEY_NOSERVERUUID
1229125527	0x4942f797	OBJECTKEY_SERVERUUIDKEY
1229125528	0x4942f798	OBJECTKEY_NOPOANAME
1229125529	0x4942f799	OBJECTKEY_SETSCID
1229125530	0x4942f79a	OBJECTKEY_SETSERVERID
1229125531	0x4942f79b	OBJECTKEY_NOUSERKEY
1229125532	0x4942f79c	OBJECTKEY_SETUSERKEY
1229125533	0x4942f79d	INVALID_INDEXED_PROFILE_1
1229125534	0x4942f79e	INVALID_INDEXED_PROFILE_2
1229125762	0x4942f882	UNSPECIFIED_MARSHAL_1
1229125763	0x4942f883	UNSPECIFIED_MARSHAL_2
1229125764	0x4942f884	UNSPECIFIED_MARSHAL_3

Table 9. Decimal minor exception codes 1299125765 to 1229125906

Decimal	Hexadecimal	Minor code reason
1229125765	0x4942f885	UNSPECIFIED_MARSHAL_4
1229125766	0x4942f886	UNSPECIFIED_MARSHAL_5
1229125767	0x4942f887	UNSPECIFIED_MARSHAL_6
1229125768	0x4942f888	UNSPECIFIED_MARSHAL_7
1229125769	0x4942f889	UNSPECIFIED_MARSHAL_8
1229125770	0x4942f88a	UNSPECIFIED_MARSHAL_9
1229125771	0x4942f88b	UNSPECIFIED_MARSHAL_10
1229125772	0x4942f88c	UNSPECIFIED_MARSHAL_11
1229125773	0x4942f88d	UNSPECIFIED_MARSHAL_12
1229125774	0x4942f88e	UNSPECIFIED_MARSHAL_13
1229125775	0x4942f88f	UNSPECIFIED_MARSHAL_14
1229125776	0x4942f890	UNSPECIFIED_MARSHAL_15

Table 9. Decimal minor exception codes 1299125765 to 1229125906 (continued)

1229125777	0x4942f891	UNSPECIFIED_MARSHAL_16
1229125778	0x4942f892	UNSPECIFIED_MARSHAL_17
1229125779	0x4942f893	UNSPECIFIED_MARSHAL_18
1229125780	0x4942f894	UNSPECIFIED_MARSHAL_19
1229125781	0x4942f895	UNSPECIFIED_MARSHAL_20
1229125782	0x4942f896	UNSPECIFIED_MARSHAL_21
1229125783	0x4942f897	UNSPECIFIED_MARSHAL_22
1229125784	0x4942f898	UNSPECIFIED_MARSHAL_23
1229125785	0x4942f899	UNSPECIFIED_MARSHAL_24
1229125786	0x4942f89a	UNSPECIFIED_MARSHAL_25
1229125787	0x4942f89b	UNSPECIFIED_MARSHAL_26
1229125788	0x4942f89c	UNSPECIFIED_MARSHAL_27
1229125789	0x4942f89d	UNSPECIFIED_MARSHAL_28
1229125790	0x4942f89e	UNSPECIFIED_MARSHAL_29
1229125791	0x4942f89f	UNSPECIFIED_MARSHAL_30
1229125792	0x4942f8a0	UNSPECIFIED_MARSHAL_31
1229125793	0x4942f8a1	UNSPECIFIED_MARSHAL_32
1229125794	0x4942f8a2	UNSPECIFIED_MARSHAL_33
1229125795	0x4942f8a3	UNSPECIFIED_MARSHAL_34
1229125796	0x4942f8a4	UNSPECIFIED_MARSHAL_35
1229125797	0x4942f8a5	UNSPECIFIED_MARSHAL_36
1229125798	0x4942f8a6	UNSPECIFIED_MARSHAL_37
1229125799	0x4942f8a7	UNSPECIFIED_MARSHAL_38
1229125800	0x4942f8a8	UNSPECIFIED_MARSHAL_39
1229125801	0x4942f8a9	UNSPECIFIED_MARSHAL_40
1229125802	0x4942f8aa	UNSPECIFIED_MARSHAL_41
1229125803	0x4942f8ab	UNSPECIFIED_MARSHAL_42
1229125804	0x4942f8ac	UNSPECIFIED_MARSHAL_43
1229125805	0x4942f8ad	UNSPECIFIED_MARSHAL_44
1229125806	0x4942f8ae	UNSPECIFIED_MARSHAL_45
1229125807	0x4942f8af	UNSPECIFIED_MARSHAL_46
1229125808	0x4942f8b0	UNSPECIFIED_MARSHAL_47
1229125809	0x4942f8b1	UNSPECIFIED_MARSHAL_48
1229125810	0x4942f8b2	UNSPECIFIED_MARSHAL_49
1229125811	0x4942f8b3	UNSPECIFIED_MARSHAL_50
1229125812	0x4942f8b4	UNSPECIFIED_MARSHAL_51
1229125813	0x4942f8b5	UNSPECIFIED_MARSHAL_52
1229125814	0x4942f8b6	UNSPECIFIED_MARSHAL_53
1229125815	0x4942f8b7	UNSPECIFIED_MARSHAL_54
1229125816	0x4942f8b8	UNSPECIFIED_MARSHAL_55
1229125817	0x4942f8b9	UNSPECIFIED_MARSHAL_56
1229125818	0x4942f8ba	UNSPECIFIED_MARSHAL_57
1229125819	0x4942f8bb	UNSPECIFIED_MARSHAL_58
1229125820	0x4942f8bc	UNSPECIFIED_MARSHAL_59
1229125821	0x4942f8bd	UNSPECIFIED_MARSHAL_60
1229125822	0x4942f8be	UNSPECIFIED_MARSHAL_61
1229125823	0x4942f8bf	UNSPECIFIED_MARSHAL_62
1229125824	0x4942f8c0	UNSPECIFIED_MARSHAL_63
1229125825	0x4942f8c1	UNSPECIFIED_MARSHAL_64
1229125826	0x4942f8c2	UNSPECIFIED_MARSHAL_65
1229125827	0x4942f8c3	UNSPECIFIED_MARSHAL_66

Table 9. Decimal minor exception codes 1299125765 to 1229125906 (continued)

1229125828	0x4942f8c4	READ_OBJECT_EXCEPTION_2
1229125841	0x4942f8d1	UNSUPPORTED_IDLTYPE
1229125842	0x4942f8d2	DSI_RESULT_EXCEPTION
1229125844	0x4942f8d4	IIOPINPUTSTREAM_GROW
1229125847	0x4942f8d7	NO_CHAR_CONVERTER_1
1229125848	0x4942f8d8	NO_CHAR_CONVERTER_2
1229125849	0x4942f8d9	CHARACTER_MALFORMED_1
1229125850	0x4942f8da	CHARACTER_MALFORMED_2
1229125851	0x4942f8db	CHARACTER_MALFORMED_3
1229125852	0x4942f8dc	CHARACTER_MALFORMED_4
1229125854	0x4942f8de	INCORRECT_CHUNK_LENGTH
1229125856	0x4942f8e0	CHUNK_OVERFLOW
1229125858	0x4942f8e2	CANNOT_GROW
1229125859	0x4942f8e3	CODESET_ALREADY_SET
1229125860	0x4942f8e4	REQUEST_CANCELLED
1229125861	0x4942f8e5	WRITE_TO_STREAM_1
1229125862	0x4942f8e6	WRITE_TO_STREAM_2
1229125863	0x4942f8e7	WRITE_TO_STREAM_3
1229125864	0x4942f8e8	WRITE_TO_STREAM_4
1229125865	0x4942f8e9	PROXY_MARSHAL_FAILURE
1229125866	0x4942f8ea	PROXY_UNMARSHAL_FAILURE
1229125867	0x4942f8eb	INVALID_START_VALUE
1229125868	0x4942f8ec	INVALID_CHUNK_STATE
1229125869	0x4942f8ed	NULL_CODEBASE_IOR
1229125889	0x4942f901	DSI_NOT_IMPLEMENTED
1229125890	0x4942f902	GETINTERFACE_NOT_IMPLEMENTED
1229125891	0x4942f903	SEND_DEFERRED_NOTIMPLEMENTED
1229125893	0x4942f905	ARGUMENTS_NOTIMPLEMENTED
1229125894	0x4942f906	RESULT_NOTIMPLEMENTED
1229125895	0x4942f907	EXCEPTIONS_NOTIMPLEMENTED
1229125896	0x4942f908	CONTEXTLIST_NOTIMPLEMENTED
1229125902	0x4942f90e	CREATE_OBJ_REF_BYTE_NOTIMPLEMENTED
1229125903	0x4942f90f	CREATE_OBJ_REF_IOR_NOTIMPLEMENTED
1229125904	0x4942f910	GET_KEY_NOTIMPLEMENTED
1229125905	0x4942f911	GET_IMPL_ID_NOTIMPLEMENTED
1229125906	0x4942f912	GET_SERVANT_NOTIMPLEMENTED

Table 10. Decimal minor exception codes 1229125907 to 1229126567

Decimal	Hexadecimal	Minor code reason
1229125907	0x4942f913	SET_ORB_NOTIMPLEMENTED
1229125908	0x4942f914	SET_ID_NOTIMPLEMENTED
1229125909	0x4942f915	GET_CLIENT_SUBCONTRACT_NOTIMPLEMENTED
1229125913	0x4942f919	CONTEXTIMPL_NOTIMPLEMENTED
1229125914	0x4942f91a	CONTEXT_NAME_NOTIMPLEMENTED
1229125915	0x4942f91b	PARENT_NOTIMPLEMENTED
1229125916	0x4942f91c	CREATE_CHILD_NOTIMPLEMENTED
1229125917	0x4942f91d	SET_ONE_VALUE_NOTIMPLEMENTED
1229125918	0x4942f91e	SET_VALUES_NOTIMPLEMENTED
1229125919	0x4942f91f	DELETE_VALUES_NOTIMPLEMENTED
1229125920	0x4942f920	GET_VALUES_NOTIMPLEMENTED
1229125922	0x4942f922	GET_CURRENT_NOTIMPLEMENTED_1

Table 10. Decimal minor exception codes 1229125907 to 1229126567 (continued)

1229125923	0x4942f923	GET_CURRENT_NOTIMPLEMENTED_2
1229125924	0x4942f924	CREATE_OPERATION_LIST_NOTIMPLEMENTED_1
1229125925	0x4942f925	CREATE_OPERATION_LIST_NOTIMPLEMENTED_2
1229125926	0x4942f926	GET_DEFAULT_CONTEXT_NOTIMPLEMENTED_1
1229125927	0x4942f927	GET_DEFAULT_CONTEXT_NOTIMPLEMENTED_2
1229125928	0x4942f928	SHUTDOWN_NOTIMPLEMENTED
1229125929	0x4942f929	WORK_PENDING_NOTIMPLEMENTED
1229125930	0x4942f92a	PERFORM_WORK_NOTIMPLEMENTED
1229125931	0x4942f92b	COPY_TK_ABSTRACT_NOTIMPLEMENTED
1229125932	0x4942f92c	PI_CLIENT_GET_POLICY_NOTIMPLEMENTED
1229125933	0x4942f92d	PI_SERVER_GET_POLICY_NOTIMPLEMENTED
1229125934	0x4942f92e	ADDRESSING_MODE_NOTIMPLEMENTED_1
1229125935	0x4942f92f	ADDRESSING_MODE_NOTIMPLEMENTED_2
1229125936	0x4942f930	SET_OBJECT_RESOLVER_NOTIMPLEMENTED
1229125937	0x4942f931	DISCONNECTED_SERVANT_1
1229125938	0x4942f932	DISCONNECTED_SERVANT_2
1229125939	0x4942f933	DISCONNECTED_SERVANT_3
1229125940	0x4942f934	DISCONNECTED_SERVANT_4
1229125941	0x4942f935	DISCONNECTED_SERVANT_5
1229125942	0x4942f936	DISCONNECTED_SERVANT_6
1229125943	0x4942f937	DISCONNECTED_SERVANT_7
1229125944	0x4942f938	GET_INTERFACE_DEF_NOT_IMPLEMENTED
1229126017	0x4942f981	MARSHAL_NO_MEMORY_1
1229126018	0x4942f982	MARSHAL_NO_MEMORY_2
1229126019	0x4942f983	MARSHAL_NO_MEMORY_3
1229126020	0x4942f984	MARSHAL_NO_MEMORY_4
1229126021	0x4942f985	MARSHAL_NO_MEMORY_5
1229126022	0x4942f986	MARSHAL_NO_MEMORY_6
1229126023	0x4942f987	MARSHAL_NO_MEMORY_7
1229126024	0x4942f988	MARSHAL_NO_MEMORY_8
1229126025	0x4942f989	MARSHAL_NO_MEMORY_9
1229126026	0x4942f98a	MARSHAL_NO_MEMORY_10
1229126027	0x4942f98b	MARSHAL_NO_MEMORY_11
1229126028	0x4942f98c	MARSHAL_NO_MEMORY_12
1229126029	0x4942f98d	MARSHAL_NO_MEMORY_13
1229126030	0x4942f98e	MARSHAL_NO_MEMORY_14
1229126031	0x4942f98f	MARSHAL_NO_MEMORY_15
1229126032	0x4942f990	MARSHAL_NO_MEMORY_16
1229126033	0x4942f991	MARSHAL_NO_MEMORY_17
1229126034	0x4942f992	MARSHAL_NO_MEMORY_18
1229126035	0x4942f993	MARSHAL_NO_MEMORY_19
1229126036	0x4942f994	MARSHAL_NO_MEMORY_20
1229126037	0x4942f995	MARSHAL_NO_MEMORY_21
1229126038	0x4942f996	MARSHAL_NO_MEMORY_22
1229126039	0x4942f997	MARSHAL_NO_MEMORY_23
1229126040	0x4942f998	MARSHAL_NO_MEMORY_24
1229126041	0x4942f999	MARSHAL_NO_MEMORY_25
1229126042	0x4942f99a	MARSHAL_NO_MEMORY_26
1229126043	0x4942f99b	MARSHAL_NO_MEMORY_27
1229126044	0x4942f99c	MARSHAL_NO_MEMORY_28
1229126045	0x4942f99d	MARSHAL_NO_MEMORY_29

Table 10. Decimal minor exception codes 1229125907 to 1229126567 (continued)

1229126046	0x4942f99e	MARSHAL_NO_MEMORY_30
1229126047	0x4942f99f	MARSHAL_NO_MEMORY_31
1229126401	0x4942fb01	RESPONSE_TIMED_OUT
1229126402	0x4942fb02	FRAGMENT_TIMED_OUT
1229126529	0x4942fb81	NO_SERVER_SC_IN_DISPATCH
1229126530	0x4942fb82	NO_SERVER_SC_IN_LOOKUP
1229126531	0x4942fb83	NO_SERVER_SC_IN_CREATE_DEFAULT_SERVER
1229126532	0x4942fb84	NO_SERVER_SC_IN_SETUP
1229126533	0x4942fb85	NO_SERVER_SC_IN_LOCATE
1229126534	0x4942fb86	NO_SERVER_SC_IN_DISCONNECT
1229126539	0x4942fb8b	ORB_CONNECT_ERROR_1
1229126540	0x4942fb8c	ORB_CONNECT_ERROR_2
1229126541	0x4942fb8d	ORB_CONNECT_ERROR_3
1229126542	0x4942fb8e	ORB_CONNECT_ERROR_4
1229126543	0x4942fb8f	ORB_CONNECT_ERROR_5
1229126544	0x4942fb90	ORB_CONNECT_ERROR_6
1229126545	0x4942fb91	ORB_CONNECT_ERROR_7
1229126546	0x4942fb92	ORB_CONNECT_ERROR_8
1229126547	0x4942fb93	ORB_CONNECT_ERROR_9
1229126548	0x4942fb94	ORB_REGISTER_1
1229126549	0x4942fb95	ORB_REGISTER_2
1229126553	0x4942fb99	ORB_REGISTER_LOCAL_1
1229126554	0x4942fb9a	ORB_REGISTER_LOCAL_2
1229126555	0x4942fb9b	LOCAL_SERVANT_LOOKUP
1229126556	0x4942fb9c	POA_LOOKUP_ERROR
1229126557	0x4942fb9d	POA_INACTIVE
1229126558	0x4942fb9e	POA_NO_SERVANT_MANAGER
1229126559	0x4942fb9f	POA_NO_DEFAULT_SERVANT
1229126560	0x4942fba0	POA_WRONG_POLICY
1229126561	0x4942fba1	FINDPOA_ERROR
1229126562	0x4942fba2	ADAPTER_ACTIVATOR_EXCEPTION
1229126563	0x4942fba3	POA_SERVANT_ACTIVATOR_LOOKUP_FAILED
1229126564	0x4942fba4	POA_BAD_SERVANT_MANAGER
1229126565	0x4942fba5	POA_SERVANT_LOCATOR_LOOKUP_FAILED
1229126566	0x4942fba6	POA_UNKNOWN_POLICY
1229126567	0x4942fba7	POA_NOT_FOUND

Table 11. Decimal minor exception codes 1229126568 to 1330446377

Decimal	Hexadecimal	Minor code reason
1229126568	0x4942fba8	SERVANT_LOOKUP
1229126569	0x4942fba9	SERVANT_IS_ACTIVE
1229126570	0x4942fbaa	SERVANT_DISPATCH
1229126571	0x4942fbab	WRONG_CLIENTSC
1229126572	0x4942fbac	WRONG_SERVERSC
1229126573	0x4942fbad	SERVANT_IS_NOT_ACTIVE
1229126574	0x4942fbae	POA_WRONG_POLICY_1
1229126575	0x4942fbaf	POA_NOCONTEXT_1
1229126576	0x4942fbb0	POA_NOCONTEXT_2
1229126577	0x4942fbb1	POA_INVALID_NAME_1
1229126578	0x4942fbb2	POA_INVALID_NAME_2
1229126579	0x4942fbb3	POA_INVALID_NAME_3

Table 11. Decimal minor exception codes 1229126568 to 1330446377 (continued)

1229126580	0x4942fbb4	POA_NOCONTEXT_FOR_PREINVOKE
1229126581	0x4942fbb5	POA_NOCONTEXT_FOR_CHECKING_PREINVOKE
1229126582	0x4942fbb6	POA_NOCONTEXT_FOR_POSTINVOKE
1229126657	0x4942fc01	LOCATE_UNKNOWN_OBJECT
1229126658	0x4942fc02	BAD_SERVER_ID_1
1229126659	0x4942fc03	BAD_SERVER_ID_2
1229126660	0x4942fc04	BAD_IMPLID
1229126665	0x4942fc09	BAD_SKELETON_1
1229126666	0x4942fc0a	BAD_SKELETON_2
1229126673	0x4942fc11	SERVANT_NOT_FOUND_1
1229126674	0x4942fc12	SERVANT_NOT_FOUND_2
1229126675	0x4942fc13	SERVANT_NOT_FOUND_3
1229126676	0x4942fc14	SERVANT_NOT_FOUND_4
1229126677	0x4942fc15	SERVANT_NOT_FOUND_5
1229126678	0x4942fc16	SERVANT_NOT_FOUND_6
1229126679	0x4942fc17	SERVANT_NOT_FOUND_7
1229126687	0x4942fc1f	SERVANT_DISCONNECTED_1
1229126688	0x4942fc20	SERVANT_DISCONNECTED_2
1229126689	0x4942fc21	NULL_SERVANT
1229126690	0x4942fc22	ADAPTER_ACTIVATOR_FAILED
1229126692	0x4942fc24	ORB_DESTROYED
1229126693	0x4942fc25	DYNANY_DESTROYED
1229127170	0x4942fe02	CONNECT_FAILURE_1
1229127171	0x4942fe03	CONNECT_FAILURE_2
1229127172	0x4942fe04	CONNECT_FAILURE_3
1229127173	0x4942fe05	CONNECT_FAILURE_4
1229127297	0x4942fe81	UNKNOWN_CORBA_EXC
1229127298	0x4942fe82	RUNTIMEEXCEPTION
1229127299	0x4942fe83	UNKNOWN_SERVER_ERROR
1229127300	0x4942fe84	UNKNOWN_DSI_SYSEX
1229127301	0x4942fe85	UNEXPECTED_CHECKED_EXCEPTION
1229127302	0x4942fe86	UNKNOWN_CREATE_EXCEPTION_RESPONSE
1229127312	0x4942fe90	UNKNOWN_PI_EXC
1229127313	0x4942fe91	UNKNOWN_PI_EXC_2
1229127314	0x4942fe92	PI_ARGS_FAILURE
1229127315	0x4942fe93	PI_EXCEPTS_FAILURE
1229127316	0x4942fe94	PI_CONTEXTS_FAILURE
1229127317	0x4942fe95	PI_OP_CONTEXT_FAILURE
1229127326	0x4942fe9e	USER_DEFINED_ERROR
1229127327	0x4942fe9f	UNKNOWN_RUNTIME_IN_BOOTSTRAP
1229127328	0x4942fea0	UNKNOWN_THROWABLE_IN_BOOTSTRAP
1229127329	0x4942fea1	UNKNOWN_RUNTIME_IN_INSAGENT
1229127330	0x4942fea2	UNKNOWN_THROWABLE_IN_INSAGENT
1229127331	0x4942fea3	UNEXPECTED_IN_PROCESSING_CLIENTSIDE_INTERCEPTOR
1229127332	0x4942fea4	UNEXPECTED_IN_PROCESSING_SERVERSIDE_INTERCEPTOR
1229127333	0x4942fea5	UNEXPECTED_PI_LOCAL_REQUEST
1229127334	0x4942fea6	UNEXPECTED_PI_LOCAL_RESPONSE
1330446336	0x4f4d0000	OMGVMCID
1330446337	0x4f4d0001	FAILURE_TO_REGISTER_OR_LOOKUP_VALUE_FACTORY
1330446338	0x4f4d0002	RID_ALREADY_DEFINED_IN_IFR
1330446339	0x4f4d0003	IN_INVOCATION_CONTEXT

Table 11. Decimal minor exception codes 1229126568 to 1330446377 (continued)

1330446340	0x4f4d0004	ORB_SHUTDOWN
1330446341	0x4f4d0005	NAME_CLASH_IN_INHERITED_CONTEXT
1330446342	0x4f4d0006	SERVANT_MANAGER_EXISTS
1330446343	0x4f4d0007	INS_BAD_SCHEME_NAME
1330446344	0x4f4d0008	INS_BAD_ADDRESS
1330446345	0x4f4d0009	INS_BAD_SCHEME_SPECIFIC_PART
1330446346	0x4f4d000a	INS_OTHER
1330446348	0x4f4d000c	POLICY_FACTORY_EXISTS
1330446350	0x4f4d000e	INVALID_PI_CALL
1330446351	0x4f4d000f	SERVICE_CONTEXT_ID_EXISTS
1330446353	0x4f4d0011	POA_DESTROYED
1330446359	0x4f4d0017	NO_TRANSMISSION_CODE
1330446362	0x4f4d001a	INVALID_SERVICE_CONTEXT
1330446363	0x4f4d001b	NULL_OBJECT_ON_REGISTER
1330446364	0x4f4d001c	INVALID_COMPONENT_ID
1330446365	0x4f4d001d	INVALID_IOR_PROFILE
1330446375	0x4f4d0027	INVALID_STREAM_FORMAT_1
1330446376	0x4f4d0028	NOT_VALUE_OUTPUT_STREAM
1330446377	0x4f4d0029	NOT_VALUE_INPUT_STREAM

If none of these steps fixes your problem, check to see if the problem has been identified and documented by looking at the available online support (hints and tips, technotes, and fixes).

For current information available from IBM Support on known problems and their resolution, see the i5/OS software page. You should also refer to this page before opening a PMR because it contains omfpr,atopm about the documents that you have to gather and send to IBM to receive help with a problem.

Chapter 26. Transactions

Troubleshooting transactions

Use this overview task to help resolve a problem that you think is related to the Transaction service.

To identify and resolve transaction-related problems, you can use the standard WebSphere Application Server RAS facilities. If you encounter a problem that you think might be related to transactions, complete the following steps:

1. Check for transaction messages in the administrative console.

   The Transaction service produces diagnostic messages prefixed by “CWWTR”. The error message indicates the nature of the problem and provides some detail. The associated message information provides an explanation and any user actions to resolve the problem.
2. Check for Transaction messages.

   Check the activity log.
3. Check for more messages in additional places.

   Check the application server’s stdout.log. For more information about a problem, check the stdout.log file for the application server, which should contain more error messages and extra details about the problem.
4. Check for messages related to the application server’s transaction log directory when the problem occurred.

   Note: If you changed the transaction log directory and a problem caused the application server to fail (with in-flight transactions) before the server was restarted properly, the server will next start with the new log directory and be unable to automatically resolve in-flight transactions that were recorded in the old log directory. To resolve this, you can copy the transaction logs to the new directory then stop and restart the application server.
5. Check the hints and tips for troubleshooting transactions.

   The “Tips for troubleshooting transactions” is an ongoing collection of troubleshooting tips, based on test and user experience. If you have suggestions for other tips, please let the IBM WebSphere writing team know.

Tips for troubleshooting transactions

This topic provides a set of tips to help you troubleshoot problems with the WebSphere Application Server transaction service.

- Peer recovery fails to acquire a lock
- “XAER_NOTA exception logged after server fails” on page 394

For messaging problems specific to WebSphere Application Server nodes, see other topics in the information center, such as “Troubleshooting WebSphere MQ messaging” on page 310, and the Application Servers support Web site.

Peer recovery fails to acquire a lock

If peer recovery of a transaction fails to acquire a file lock that is needed to perform recovery processing, you should see the following messages:

```
[10/26/04 8:41:38:887 CDT] 00000029 CoordinationL A   CWWTR0100_GENERIC_ERROR
[10/26/04 8:41:39:100 CDT] 00000029 RecoveryHandl A   CWWTR0100E: An attempt to
acquire a file lock need to perform recovery processing failed. Either the target
server is active or the recovery log configuration is incorrect
....
[10/26/04 8:42:34:921 CDT] 00000027 HAGroupImpl   I   CWRHA0130I: The local member
```

```
of group GN_PS=fwsitkaCell01\fwwsaix1Node01\GriffinServer3,IBM_hc=GriffinCluster,type
=WAS_TRANSACTIONS has indicated that is it not alive. The JVM will be terminated.
[10/26/04 8:42:34:927 CDT] 00000027 SystemOut     O Panic:component requested panic
from isAlive
```

To troubleshoot the cause of failure to acquire the file lock, check the following factors:
- If you have enabled failover of transaction log recovery on the server cluster and are using a NAS devise for the transaction logs, check that the DFS level on your machine is at a correct level for the NAS DFS level. If the two levels are not correct, the transaction logs cannot be accessed.
- If you are running as non-root, check that the id numbers of the non-root user and group match on all machines involved with peer recovery.
- If you have a policy defined for transaction, review the policy to ensure that you are giving control to the correct servers (perhaps you need to add to or reorder the preferred server list).

XAER_NOTA exception logged after server fails

If an application server fails, and the end transaction record is not forced to disk immediately, you may or may not recover a transaction.

WebSphere Application Server does not force the end record to the log, so it is up to the operating system/network file system to decide when to write to the disk. The record would be forced if the server was shutdown cleanly. The transaction service is designed to cope with the case of the end record never being written to disk - when it gets an XAER_NOTA returned from the databases.

```
[date time] 00000057 WSRdbXaResour E   CWWRA0302E:  XAException occurred.
                                        Error code is: XAER_NOTA (-4).  Exception is: XAER_NOTA
```

If there is a transaction without an end record left in the transaction log, the transaction service tries to check with the database. If the transaction has completed, the database indicates that there is nothing to complete (XAER_NOTA). This is normal behavior, and not an error.

Transaction service exceptions

This topic lists the exceptions that can be thrown by the WebSphere Application Server transaction service. The exceptions are listed in the following groups:
- Standard exceptions
- Heuristic exceptions

If the EJB container catches a system exception from the business method of an enterprise bean, and the method is running within a container-managed transaction, the container rolls back the transaction before passing the exception on to the client. For more information about how the container handles the exceptions thrown by the business methods for beans with container-managed transaction demarcation, see the section *Exception handling* in the Enterprise JavaBeans 2.0 specification. That section specifies the container's action as a function of the condition under which the business method executes and the exception thrown by the business method. It also illustrates the exception that the client receives and how the client can recover from the exception.

Standard exceptions

The standard exceptions such as TransactionRequiredException, TransactionRolledbackException, and InvalidTransactionException are defined in the Java Transaction API (JTA) 1.0.1 Specification.

InvalidTransactionException

This exception indicates that the request carried an invalid transaction context.

TransactionRequiredException exception

This exception indicates that a request carried a null transaction context, but the target object requires an active transaction.

TransactionRolledbackException exception

This exception indicates that the transaction associated with processing of the request has been rolled back, or marked for roll back. Thus the requested operation either could not be performed or was not performed because further computation on behalf of the transaction would be fruitless.

Heuristic exceptions

A heuristic decision is a unilateral decision made by one or more participants in a transaction to commit or rollback updates without first obtaining the consensus outcome determined by the Transaction Service. Heuristic decisions are an issue only after the participant has been prepared and the second phase of commit processing is underway. Heuristic decisions are normally made only in unusual circumstances, such as repeated failures by the transaction manager to communicate with a resource manage during two-phase commit. If a heuristic decision is taken, there is a risk that the decision differs from the consensus outcome, resulting in a loss of data integrity.

The following list provides a summary of the heuristic exceptions. For more detail, see the Java Transaction API (JTA) 1.0.1 Specification.

HeuristicRollback exception

This exception is raised on the commit operation to report that a heuristic decision was made and that all relevant updates have been rolled back.

HeuristicMixed exception

This exception is raised on the commit operation to report that a heuristic decision was made and that some relevant updates have been committed and others have been rolled back.

Exceptions thrown for transactions involving both single- and two-phase commit resources

The exceptions that can be thrown by transactions that involve single- and two-phase commit resources are the same as those that can be thrown by transactions involving only two-phase commit resources.

The exceptions that can be thrown are listed in the Reference: Generated API documentation found in the Reference section of the WebSphere Application Server Version 6.1 Information Center.

Chapter 27. Learn about WebSphere programming extensions

Use this section as a starting point to investigate the WebSphere programming model extensions for enhancing your application development and deployment.

See Learn about WebSphere applications: Overview and new features for a brief description of each WebSphere extension.

See the *Developing and deploying applications* PDF book for a brief description of each WebSphere extension.

In addition, now your applications can use the Eclipse extension framework. Your applications are extensible as soon as you define an extension point and provide the extension processing code for the extensible area of the application. You can also plug an application into another extensible application by defining an extension that adheres to the target extension point requirements. The extension point can find the newly added extension dynamically and the new function is seamlessly integrated in the existing application. It works on a cross Java 2 Platform, Enterprise Edition (J2EE) module basis.

The application extension registry uses the Eclipse plug-in descriptor format and application programming interfaces (APIs) as the standard extensibility mechanism for WebSphere applications. Developers that build WebSphere application modules can use WebSphere Application Server extensions to implement Eclipse tools and to provide plug-in modules to contribute functionality such as actions, tasks, menu items, and links at predefined extension points in the WebSphere application. Read the information about Application extension registry in the *Developing and deploying applications* PDF book.

ActivitySessions

Troubleshooting ActivitySessions

Use this overview task to help resolve a problem that you think is related to the ActivitySession service.

To identify and resolve ActivitySession-related problems, you can use the standard WebSphere Application Server RAS facilities. If you encounter a problem that you think might be related to ActivitySessions, complete the following stages:

1. Check for ActivitySession messages in the admin console. The ActivitySession service produces diagnostic messages prefixed by “WACS”. The error message indicates the nature of the problem and provides some detail. The associated message information provides an explanation and any user actions to resolve the problem.
2. Check for ActivitySession messages. The ActivitySession service produces diagnostic messages prefixed by “WACS”. The error message indicates the nature of the problem and provides some detail. The associated message information provides an explanation and any user actions to resolve the problem. Activity log messages produced by the ActivitySession service are accompanied by Log and Trace Analyzer descriptions.

   Check in the application server’s SystemOut log at *was_home*\logs*server*\SystemOut for error messages with the prefix WACS. If needed, check other messages, which should provide extra details about the problem.

Application profiling

Application profiling exceptions

The following exceptions are thrown in response to various illegal actions related to application profiling.

com.ibm.ws.exception.RuntimeWarning

This exception is thrown when the application is started, if the application is configured incorrectly. The startup is consequently terminated. Some examples of bad configurations include:

- A task configured on two different application profiles.
- A method configured with two different task run-as policies .

com.ibm.websphere.appprofile.IllegalTaskNameException

This exception is raised if an application attempts to programmatically set a task when that task has not been configured as a task name reference.

Dynamic cache

Troubleshooting the dynamic cache service

This topic includes steps to troubleshoot the dynamic cache service.

Complete the steps below to resolve problems that you think are related to the dynamic cache service.

1. Review the logs. Messages that are prefaced with *DYNA* result from dynamic cache service operations.
   a. Find any messages prefaced with *DYNA* in the log you are viewing, and write down the message IDs. A sample message having the message ID *DYNA0030E* follows:

      ```
      DYNA0030E: "property" element is missing required attribute "name".
      ```

   b. Find the message for each message ID in the information center for WebSphere Application Server. In the information center navigation tree, click ***product_name*** **> Reference > Troubleshooter > Messages > DYNA** to view dynamic cache service messages.
   c. Read the message **Explanation** and **User Action** statements. A search for the message ID `DYNA0030E` displays a page having the following message:

```
DYNA0030E: "property" element is missing required attribute "name".
  Explanation: A required attribute was missing in the cache configuration.
  User Action: Add the required attribute to your cache configuration file.
```

      This explanation and user action suggests that you can fix the problem by adding or correcting a required attribute in the cache configuration file.

   d. Try the solutions stated under **User Action** in the *DYNA* messages.
2. Use the cache monitor to determine whether the dynamic cache service is functioning as expected. The cache monitor is an installable Web application that displays simple cache statistics, cache entries, and cache policy information. Read the information about cache monitor in the *Setting up the application serving environment* PDF book.
3. If you have completed the preceding steps and still cannot resolve the problem, contact your IBM software support representative.

   The IBM representative might ask you to complete a diagnostic trace. To enable tracing in the administrative console, click **Troubleshooting > Logs and trace > *server_name* > Diagnostic trace** and specify **Enable trace with the following specification**. The IBM representative can tell you what trace specification to enter. Note that dynamic cache trace files can become large in a short period of time; you can limit the size of the trace file by starting the trace, immediately recreating the problem, and immediately stopping the trace.

   For current information available from IBM Support on known problems and their resolution, see the IBM Support page.

   For technical support on dynamic cache service, see the IBM Support page.

Troubleshooting tips for the dynamic cache service

The dynamic cache service works within an application server Java virtual machine (JVM), intercepting calls to cacheable objects. This article describes some common runtime and configuration problems and remedies.

Servlets are not cached

Recommended response Enable servlet caching. On the Web container settings page of the administrative console, select the **Enable servlet caching** check box. Read the information about the Web container settings in the *Developing and deploying applications* PDF book.

Cache entries are not written to disk

Explanation Cache entries are written to disk when the cache is full and new entries are added to the memory cache. Cache entries also are written to disk when **Flush to disk** is enabled in the administrative console and the server is stopped.

Recommended response Verify that **Disk offload** is enabled on the dynamic cache service settings page of the administrative console. Also verify that cache entries written to disk are serializable and do not have the `PersistToDisk` configuration set to `false`. Read more information about the dynamic cache settings in the *Developing and deploying applications* PDF book.

Some servlets are not replicated or written to disk

Recommended response Ensure that the attributes and response are serializable. If you do not want to store the attributes, use the following property in your cache policy:

```
<property name="save-attributes">false</property>
```

Dynamic cache service does not cache fragments on the Edge

Recommended response Set the `EdgeCacheable` property to `true` in the cache policy for those entries that are to be cached on the Edge.

```
<property name="EdgeCacheable">true</property>
```

Dynamic cache invalidations are not sent to the IBM HTTP Server plug-in

Explanation The `DynaCacheEsi.ear` file is required to send invalidations to external caches.

Recommended response Install `DynaCacheEsi.ear` using the administrative console. See the *Developing and deploying applications* PDF for more information.

Cache entries are evicted often

Problem The cache is full and new entries are added to the cache.

Explanation Cache entries are evicted when the cache is full and new entries are added to the cache. A least recently used (LRU) eviction mechanism removes the least recently used entry to make space for the new entries.

Recommended response Enable **Disk offload** on the Dynamic cache service settings page of the administrative console so the entries are written to disk. You can also increase the cache size to accommodate more entries in the cache. Read more information about the dynamic cache settings in the *Developing and deploying applications* PDF book.

Cache entries in disk with timeout set to 0 expire after one day

Explanation The maximum lifetime of an entry in disk cache is 24 hours. A timeout of `0` in the cache policy configures these entries to stay in disk cache for one whole day, unless they are evicted earlier.

Recommended response Set the timeout for the cache policy to a number less than `0`.

I cannot monitor cache entries on the Edge

Explanation Use the dynamic cache monitor to monitor the contents of the memory cache, disk cache and external caches, like the Edge cache. For the ESI processor's cache to be visible in the cache monitor, the `DynaCacheEsi.ear` application must be installed and the `esiInvalidationMonitor` property must be set to `true` in the `plugin-cfg.xml` file.

Recommended response Set the `esiInvalidationMonitor` property in the `plugin-cfg.xml` file to `true`. See the *Administering applications and their environment* PDF for more information.

I want to tune cache for my environment

Recommended response Use the Tivoli Performance viewer to study the caching behavior for your applications. Also consider performing the following actions:

- Increase the priority of cache entries that are expensive to regenerate.
- Modify timeout of entries so that they stay in memory as long as they are valid.
- Enable disk offload to store LRU evicted entries.
- Increase the cache size.

Cleaning the disk cache files after installing the fix pack or a new release if you use the disk cache function

Symptom If the server is configured to use the disk cache, you must delete the disk cache files because the disk cache files are not compatible to the previous version.

Problem Failure to remove the old disk cache files results in a ClassCastException error in the systemerr.log file when you access the cache from the disk.

Recommended response To delete the disk cache, perform the following steps:

1. Note your disk offload location. If you do not know the disk cache offload location, perform the following steps:
   a. Click **Servers > Application servers >** *server_name* **> Container services > Dynamic cache service** in the administrative console navigation tree.
   b. The location is specified in the Disk offload field. If the location is not specified, the default directory *profile_root*`/temp/`*your_node*`/`*server_name*`/_dynacache` is used.
2. Make sure that the you stop the server and delete all the files under the offload location.
3. If you use the Network Deployment product, delete the disk cache files for each server.

Setting the flush attribute to true on every <jsp:include>tag in the cacheable JavaServer Pages file

Symptom When you obtain the JavaServer Pages (JSP) file from the dynamic cache, a part of the page is not displayed.

Problem The flush attribute is set to false on the `<jsp: include>` tag in the JSP file.

Description When the cacheable JSP file includes another JSP file and if the flush attribute is set to false on the `<jsp: include>` tag, any data written to the parent output stream before the`<jsp: include>` tag are not cached.

Recommended response Set `flush=true` on every `<jsp: include>` tag in the cacheable JSP file.

Dynamic cache limitation when using the JSTL <c:import> tag to include a fragment

Problem When a cacheable fragment is included using the JavaServer Pages Standard Tag Library (JSTL) `<c:import>` tag, part of the page content disappears and part of the page content displays twice on a cache hit.

Description Dynamic cache relies on flushing the content before and after including a fragment so that the parent content before the include is not lost, and also to prevent pulling the child content into the parent fragment.

However, in the case of the JSTL `<c:import>` tag, the flush=true attribute, which flushes the parent writer before the child fragment is actually invoked, is not supported. Also, JSTL buffers the responses, so that the child writer is not flushed right after the child fragment is done. Subsequently, the child response is pulled into the parent.

Recommended response To avoid this problem, surround the `<c:import>` statement with out.flush method statements as follows:

```
<% out.flush(); %>
<c:import url="DNCParent2.jsp" />
<% out.flush(); %>
```

Internationalization

Internationalization service errors

Certain conditions might cause the internationalization service not to start, to issue java.lang.IllegalStateException exceptions while an application is running, or to exercise default behaviors.

The java.lang.IllegalStateException exception indicates one of the following things:

- An application component attempted an operation that is not supported by the internationalization programming model.

  The IllegalStateException exception is issued whenever a server application component whose internationalization type is set to container-managed internationalization (CMI) attempts to set invocation context. This behavior is a violation of the CMI policy, under which servlets and enterprise beans cannot modify their invocation internationalization context.
- An anomaly occurred that disabled the service.

  For instance, if the internationalization service is not properly initialized, the Java Naming and Directory Interface (JNDI) lookup on the UserInternationalization URL attribute issues a javax.naming.NameNotFoundException exception that contains an IllegalStateException instance.

The following conditions can occur while your internationalized application is running. These conditions might cause the internationalization service not to start, to issue IllegalStateException exceptions, or to exercise default behaviors:

- “The service is disabled ” on page 402
- “The service is not started” on page 402
- “Invalid context element” on page 403
- “Missing context element” on page 403
- “Invalid policy” on page 403
- “Missing policy” on page 403

If you encounter unexpected or exceptional behavior, the problem is likely related to one of these conditions. You need to examine the trace log to investigate these conditions, which requires that you configure the diagnostic trace service to generate messages about internationalization service function. To get started with logging and tracing, see “Tracing and logging configuration” on page 112.

The trace strings for the internationalization service follow; use both:
`com.ibm.ws.i18n.context.*=all=enabled:com.ibm.websphere.i18n.context.*=all=enabled`

The service is disabled

The internationalization service is not initialized when the startup setting is cleared. The service generates a message that indicates whether it is enabled or disabled. Applications cannot access the internationalization API when the service is disabled. If an application attempts a JNDI lookup to obtain the UserInternationalization reference, the lookup fails with a NamingException exception, indicating that the reference cannot be found. In addition, the service does not scope (propagate) internationalization context on incoming (outgoing) business method calls.

The service is not started

The internationalization service is operational whenever it is in the STARTED state. For example, if an application attempts to access internationalization context and the service is not started, the API issues an IllegalStateException exception. In addition, the service does not provide runtime support for servlets and enterprise beans.

As an application server progresses through its life cycle, it initializes, starts, stops, and terminates (destroys) the internationalization service. If an anomaly occurs during initialization, the service does not start. After the service is started, its state can change to BLOCKED in the event that a serious error occurs. The service generates a message for every state change.

If a trace message indicates that the service is not STARTED, examine previous messages to determine the problem. For instance, the internationalization service does not start if the activity service is unavailable and a message is displayed to that effect during initialization of the internationalization service.

During startup, the following messages indicate potential configuration or runtime problems:

No ORB support
: The service cannot obtain an instance of the object request broker (ORB). This condition is a fatal error. Examine the SystemErr.log and SystemOut.log files for information.

No TCM support
: The service cannot obtain an instance of its thread context manager (TCM). This condition is a fatal error. Examine the SystemErr.log and SystemOut.log files for information.

No IIOP (activity service) support
: The service cannot register with the activity service. This condition is a fatal error. The internationalization service cannot propagate or receive context on Internet Inter-ORB Protocol (IIOP) requests without activity service support. Examine the SystemErr.log and SystemOut.log files for information.

No AsynchBeans support
: The service cannot register into the asynchronous beans environment. This warning indicates that the asynchronous beans environment cannot support internationalization context.

No EJB container support
: The service cannot register with the Enterprise JavaBeans (EJB) container. This warning indicates that the internationalization service cannot support enterprise beans. Without EJB container support, internationalization contexts do not scope properly to EJB business methods. Review the trace log for any EJB container-related error conditions.

No Web container support
: The service cannot register with the Web container. This warning indicates that the internationalization service cannot support servlets and JavaServer Page (JSP) files. Without Web container support, internationalization contexts do not scope properly to servlet service methods. Review the trace log for any Web container-related error conditions.

No Metadata support
: The service cannot register with the metadata service. This warning indicates that the internationalization service cannot process the internationalization policies within application

deployment descriptors. Without metadata support, the service associates the default internationalization context management policy, [CMI, RunAsCaller], to every servlet lifecycle method and enterprise bean business method invocation. Review the trace log for any metadata service-related error conditions.

No JNDI (Naming service) support
The service cannot bind the UserInternationalization object into the namespace. This condition is a fatal error. Application components are unable to access internationalization context API references, and are therefore unable to access internationalization context elements. Review the trace log for any Naming (JNDI) service-related error conditions.

No API support
The service cannot obtain an instance of an internationalization context API object. This condition is a fatal error. Application components are unable to access internationalization context API references, and are therefore unable to access internationalization context elements.

Invalid context element

The service detected an invalid internationalization context element. For example, the internationalization service does not support TimeZone instances of a type other than java.util.SimpleTimeZone. If the service encounters an unusable element, it logs a message and substitutes the corresponding default element of the JVM.

Missing context element

The service detected a missing internationalization context element. Incoming requests (for example, from application servers that do not support the internationalization service) lack internationalization context. When the service attempts to access a caller internationalization context element (which does not exist in this case), the service logs a message and substitutes the corresponding default element of the Java virtual machine (JVM).

Whenever possible, enable the internationalization service within all clients and hosting application servers that comprise an internationalized enterprise application. Read more information about Administering the internationalization service in the *Administering applications and their environment* PDF book.

Invalid policy

The internationalization service detected a malformed internationalization policy in the application deployment descriptor. The service replaces the malformed attribute with the appropriate default. For instance, if the internationalization type for an entity bean is set to `Application` during the run of a servlet or EJB business method call, the service logs the inconsistency and enforces the `Container` setting instead.

Also, AMI application components do have an implicit container internationalization attribute. By default they run as server. The service silently enforces the implicit policy, `[AMI, RunAsServer]`, and logs messages to this effect.

Invalid container internationalization attributes are likely to occur when specifying the Locales and Time zone ID fields. When encountering invalid locales and time zone IDs within attributes, the service replaces each value with the corresponding default element of the JVM. Be sure to follow the guidelines provided in the *Developing and deploying applications* PDF book.

Missing policy

The service detected a missing internationalization policy. The service replaces the missing policy with the appropriate default. For instance, if the internationalization type is missing for a servlet or enterprise bean, the service sets the attribute to Container.

Container internationalization attributes are not mandatory for CMI application components. In the event that a CMI servlet or EJB business method lacks a container internationalization attribute, the service silently enforces the implicit policy `[CMI, RunAsCaller]`.

When an application lacks internationalization policies in its deployment descriptor, or metadata support is unavailable, the service logs a message and applies the policy `[CMI, RunAsCaller]` on every servlet service method and EJB business method invocation.

Read the information in the *Developing and deploying applications* PDF book:
- Assembling internationalized applications
- Container internationalization attributes
- Internationalization type

Appendix. Directory conventions

References in product information to *app_server_root*, *profile_root*, and other directories infer specific default directory locations. This topic describes the conventions in use for WebSphere Application Server.

These file paths are default locations. You can install the product and other components in any directory where you have write access. You can create profiles in any valid directory where you have write access. Multiple installations of WebSphere Application Server products or components, of course, require multiple locations.

app_server_root - the install_root for WebSphere Application Server
The default installation root directory for WebSphere Application Server is the /QIBM/ProdData/WebSphere/AppServer/V61/Base directory.

profile_root
The default directory for a profile named *profile_name* for WebSphere Application Server is the /QIBM/UserData/WebSphere/AppServer/V61/Base/profiles/*profile_name* directory.

app_server_user_data_root - the user_data_root for WebSphere Application Server
The default user data directory for WebSphere Application Server is the /QIBM/UserData/WebSphere/AppServer/V61/Base directory.

plugins_root
The default installation root directory for Web server plug-ins is the /QIBM/ProdData/WebSphere/Plugins/V61/webserver directory.

plugins_user_data_root
The default Web server plug-ins user data root is the /QIBM/UserData/WebSphere/Plugins/V61/webserver directory.

plugins_profile_root
The default Web server plug-ins profile root is the /QIBM/UserData/WebSphere/Plugins/V61/webserver/profiles/*profile_name* directory.

app_client_root
The default installation root directory for the J2EE WebSphere Application Client is the /QIBM/ProdData/WebSphere/AppClient/V61/client directory.

app_client_user_data_root
The default J2EE WebSphere Application Client user data root is the /QIBM/UserData/WebSphere/AppClient/V61/client directory.

app_client_profile_root
The default J2EE WebSphere Application Client profile root is the /QIBM/UserData/WebSphere/AppClient/V61/client/profiles/*profile_name* directory.

web_server_root
The default web server path is /www/*web_server_name*.

shared_product_library
The shared product library, which contains all of the objects shared by all Version 6.1 installations on the system, is QWAS61. This library contains objects such as the product definition, the subsystem description, the job description, and the job queue.

product_library
The product library, which contains program and service program objects (similar to .exe, .dll, .so objects for Windows, Linux, and UNIX operating system platforms), is: QWAS61*x* where *x* is A, B, C, ... For the default installation, the value is QWAS61A.

product_lib
The product library that contains the service program objects for the web server plugins. For the

default Web Server Plugins install, this is QWAS61A. If you install the Web Server Plugins multiple times, the product_lib is QWAS61c, where *c* is B, C, D, ... The plugins_install_root/properties/product.properties contains the value for the product library..

cip_app_server_root
The default installation root directory is the /QIBM/ProdData/WebSphere/AppServer/V61/Base/cip/*cip_uid* directory for a customized installation package (CIP) produced by the Installation Factory.

A CIP is a WebSphere Application Server product bundled with optional maintenance packages, an optional configuration archive, one or more optional enterprise archive files, and other optional files and scripts.

cip_user_data_root
The default user data root directory is the /QIBM/UserData/WebSphere/AppServer/V61/Base/cip/*cip_uid* directory for a customized installation package (CIP) produced by the Installation Factory.

cip_profile_root
The default profile root directory is the /QIBM/UserData/WebSphere/AppServer/V61/Base/cip/*cip_uid*/profiles/*profile_name* directory for a customized installation package (CIP) produced by the Installation Factory.

Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, is the user's responsibility.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Intellectual Property & Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
USA

Trademarks and service marks

For trademark attribution, visit the IBM Terms of Use Web site (http://www.ibm.com/legal/us/).